Escuela Técnica Superior de Ingeniería Infomática

Grado en Ingeniería Informática – Ingeniería del Software

TRABAJO FIN DE GRADO

Detección de Fake News mediante técnicas de Deep Learning

Autor/es:

Juan Carlos Alonso Valenzuela

47548996-S

Tutor/es:

Sergio Segura Rueda

Primera Convocatoria

Curso 2019/2020

Resumen

En el **contexto** de los medios de comunicación, las noticias falsas (también conocidas como fake news), son un tipo de noticia diseñada y emitida con la intención deliberada de engañar, inducir a error, manipular decisiones personales o desprestigiar a una determinada entidad o persona. Al presentar hechos falsos como si fuesen reales, estas noticias constituyen una gran amenaza para la credibilidad de los medios serios y los periodistas profesionales.

El **objetivo** de este proyecto es desarrollar un sistema de detección de fake news aplicando diversos métodos de clasificación y realizando un estudio comparativo entre ellos, para así determinar cuál es el más apropiado para resolver el problema, indicando siempre las ventajas e inconvenientes de cada uno.

El **método** empleado para alcanzar el objetivo del proyecto ha consistido en aplicar un total de nueve modelos que emplean las últimos avances en el campo del procesamiento del lenguaje natural y las redes neuronales. Para realizar la clasificación, estas redes han sido entrenadas utilizando el TI-CNN dataset [1], creado por un equipo de investigadores compuesto por miembros de las universidades de Pekín (Beihang University), Chicago (University of Illinois) y Florida (Florida State University), y que ya ha sido empleado en varias publicaciones científicas. Además de aplicar modelos neuronales avanzados, también se emplearán otros métodos más simples que permitirán obtener una mayor comprensión del conjunto de datos y realizar una comparación con los resultados devueltos con respecto a las redes más complejas, cumpliendo la función de baseline.

Para la implementación de estos algoritmos se han utilizado algunas de las librerías más extendidas en el campo del Machine Learning, como TensorFlow, scikit-learn, nltk, numpy y pandas, entre otras.

Como **resultado** de este proyecto, se ha logrado obtener un F1-Score del 96,66%, tras realizar la optimización de los hiperparámetros de la red y preprocesar los datos. Estos resultados superan a los de la publicación original, que obtuvo un F1-Score del 92,1%. La aplicación de métodos más simples, además de cumplir la función de baseline, ha servido para detectar sesgos y limitaciones en el conjunto de datos que explican hasta cierto punto los buenos resultados obtenidos por la publicación original.

Como **conclusión**, se han cumplido todos los objetivos del proyecto con éxito, además de mejorar los resultados de la publicación original. El empleo de la metodología Scrum para llevar a cabo el proyecto ha sido un factor determinante para la finalización de este dentro de los plazos acordados y ha permitido organizar las tareas a realizar en sprints. Gracias a la realización del proyecto, se han adquirido conocimientos de análisis de datos, Machine Learning, Deep Learning y procesamiento del lenguaje natural que son muy demandados en la actualidad y pueden aplicarse para resolver una gran cantidad de problemas. También se ha obtenido experiencia práctica en labores de investigación, ya que para la consecución del proyecto ha sido necesario leer y analizar una gran cantidad de artículos científicos.

Contenido

Índice de figuras

Índice de tablas

Índice de ecuaciones

PARTE I. INTRODUCCIÓN Y PLANIFICACIÓN

1. Introducción

1.1. Motivación

Las noticias falsas, popularmente conocidas por el término en inglés fake news, son un tipo de bulo consistente en un contenido pseudoperiodístico que presenta hechos falsos como si fueran reales a través de distintos medios, tales como portales de noticas, prensa escrita, radio, televisión y redes sociales. El principal objetivo de esta clase de noticias es la desinformación.

Las razones por las que pueden surgir las fake news son muy variadas, pueden ser creadas por causas políticas con el objetivo de desprestigiar a una determinada ideología o partido, para generar ganancias económicas o incluso por motivos religiosos. Otro tipo de noticias que podrían clasificarse como fake news son las conocidas como clickbait [2], caracterizadas por emplear títulos o miniaturas sensacionalistas o engañosas con la intención de generar visitas y obtener ingresos por publicidad.

La creación y difusión de noticias falsas existe desde la antigüedad, aunque en la actualidad se ha vuelto mucho más preocupante a causa del incremento del uso de Internet en las últimas décadas, ya que la generación de información y la capacidad de compartirla a nivel global se ha incrementado considerablemente, pero esto no ha ocurrido con la capacidad de contrastar las fuentes o la credibilidad de dicha información.

A menudo estas noticias se difunden más rápido que las noticias reales (a causa seguramente de su carácter sensacionalista) de manera que suele ser demasiado tarde para cuando se desmiente la noticia, porque el daño ya está hecho y la noticia desmentida no suele llegar a tanta gente al resultar generalmente menos interesante.

Esta práctica supone una amenaza para la credibilidad de los medios serios y profesionales porque el flujo constante de información aleja al periodismo del rigor informativo y la verificación, en muchas ocasiones puede ocurrir que salga mejor beneficiado el periodista que publica una noticia en cuanto obtiene información publicable que aquel que solo la publica una vez haya contrastado y verificado sus fuentes.

Las fake news han alcanzado una mayor notoriedad tras las elecciones a la presidencia de los Estados Unidos de 2016, a causa del descubrimiento de que esta clase de noticias fueron cruciales para el triunfo de Donald Trump [3].

La preocupación por la influencia de las fake news en algunos momentos políticos clave, ha llevado a muchos líderes mundiales a impulsar legislaciones para el control de la difusión de información falsa en redes sociales. Un ejemplo de esto es la ley impulsada por Angela Merkel para aplicar multas a las plataformas que no eliminen mensajes de odio, xenofobia o noticias falsas en un plazo de 24 horas. Asimismo, también se están realizando numerosas acciones que buscan que las personas incrementen su nivel de atención y discernimiento sobre las noticias que reciben, con el fin de detectar las fake news y evitar que sean divulgadas.

Estas soluciones son, en la mayoría de los casos, insuficientes para evitar la creación y difusión de fake news, porque estas se divulgan con gran rapidez (una noticia puede volverse viral y llegar a millones de personas en bastante menos de 24 horas) y muchas veces son muy difíciles de detectar, incluso para gente que esté concienciada.

En los últimos años, se han producido una gran cantidad de avances en Inteligencia Artificial, hasta el punto de que cada vez está más presente en nuestro día a día. Una de las ramas de la Inteligencia Artificial con mayor auge en los últimos años es el procesamiento del lenguaje natural, que se basa en la idea de que una máquina pueda analizar y comprender textos escritos en lenguaje natural conociendo no solamente las palabras que componen el texto, sino también el contexto en el que aparecen, esto tiene una gran cantidad de aplicaciones, como generación de textos, traducción automática y clasificación de documentos.

Por estas razones, se ha decidido aplicar algunas de las últimas técnicas relativas a procesamiento del lenguaje natural y aprendizaje profundo para la identificación automática de fake news, con la intención de detectar patrones característicos de estas.

1.2. Objetivos del proyecto

1.2.1. Objetivos docentes

A continuación, se enumeran los objetivos docentes que pretenden alcanzarse durante la realización del presente proyecto:

- Aplicar técnicas de Procesamiento del Lenguaje Natural, Procesamiento de datos y Deep Learning a la solución de un problema real de actualidad. Los conocimientos de este dominio serán adquiridos a lo largo del proyecto, con la intención de servir como base en una futura carrera en el campo de Data Science.
- Ser capaz de comparar y contrastar los resultados de la aplicación de diferentes algoritmos de Procesamiento del Lenguaje Natural sobre un conjunto de datos, comparando las ventajas e inconvenientes de cada uno de estos métodos, justificando siempre la elección de uno sobre otro.
- Aprender a emplear tecnologías relacionadas con el dominio del problema, tales como pandas, sklearn, nltk, keras, numpy, matplotlib o TensorFlow. Estas tecnologías son explicadas con mayor detalle en la sección Tecnologías de esta memoria.
- Emplear metodologías ágiles de manera práctica para la correcta gestión y planificación de un proyecto informático de escala media, realizando además estimaciones de tiempo y costes.
- Redactar una memoria con una estructura definida que explique los métodos empleados para resolver los problemas propuestos, además de la metodología seguida, justificando siempre las decisiones tomadas. La estructura del documento es desglosada en la sección Estructura del documento.

1.2.2. Objetivos técnicos

Obtener un algoritmo que pueda distinguir con éxito las noticias legítimas de las conocidas como fake news de manera automática, alcanzando resultados de predicción fiables.

1.3. Estructura del documento

En esta memoria se realizará un seguimiento del proyecto desde su inicio hasta su final, documentando en su transcurso las diferentes etapas o fases de la gestión del proyecto.

A continuación, se muestran los puntos que tratan cada una de las partes del presente documento:

PARTE I. INTRODUCCIÓN Y PLANIFICACIÓN

En esta parte se exponen las motivaciones para la realización del proyecto, así como los objetivos docentes y técnicos que se pretenden alcanzar. También incluye un glosario de abreviaturas y acrónimos usados en la memoria, además de la planificación inicial de tiempo y de costes del proyecto.

PARTE II. MATERIAS RELACIONADAS

Se realizará un estudio de posibles metodologías a seguir durante el desarrollo de un proyecto informático, con el objetivo de escoger aquella que se adapte mejor a este proyecto. En la segunda sección de esta parte, se desglosarán las tecnologías y herramientas utilizadas.

PARTE III. SISTEMA DESARROLLADO

Compuesto por una explicación del diseño inicial del proyecto, con sus objetivos, requisitos y arquitectura, seguido de la memoria de cada una de las iteraciones del desarrollo del proyecto.

PARTE IV. MANUALES

Manual de usuario para poder ejecutar correctamente el proyecto.

PARTE V. CONCLUSIONESF

Análisis de los resultados finales del proyecto en cuanto a desviaciones con respecto a la planificación temporal y de costes inicial realizada en la PARTE II. MATERIAS RELACIONADAS, además de las lecciones aprendidas. Por último, se concluye si se han alcanzado los objetivos iniciales del proyecto y se proponen posibles mejoras a realizar.

1.4. Abreviaturas y acrónimos utilizados

- **ASD:** Adaptative Software Development
- **BOE:** Boletín Oficial del Estado
- **BOW:** Bag of Words
- **CBOW:** Continuous Bag of Words
- **CRC:** Clase-Responsabilidad-Colaborador
- **DBOW:** Distributed Bag of Words
- **DF:** Document Frequency
- **DL:** Deep Learning
- **DM:** Distributed Memory
- **DS:** Data Scientist
- **FDD:** Feature-Driven Development
- **FOGASA:** Fondo de Garantía Social
- **IDF:** Inverse Document Frequency
- **IMS:** Incapacidad Permanente, Muerte o Supervivencia
- **IT:** Incapacidad Temporal
- **JP:** Jefe de Proyecto
- **LSTM:** Long Short Term Memory
- **Mb:** Megabyte
- **NLP:** Natural Language Processing
- **nltk:** Natural Language Toolkit
- **POS:** Part-Of-Speech
- **sklearn:** Scikit Learn
- **TF:** Term Frequency
- **TF-IDF:** Term Frequency – Inverse Document Frequency
- **XP:** eXtreme Programming

2. Planificación

En esta sección se aborda la Planificación del proyecto, tanto temporal como de costes. Realizar una correcta planificación temporal y de costes desde el inicio del desarrollo de un proyecto es un factor determinante para el éxito de este. Para los proyectos será vital que el análisis estimado de tiempo y coste sea lo más cercano al real, para evitar realizar las entregas fuera de plazo o exceder el presupuesto establecido.

2.1. Planificación temporal

El objetivo de la planificación temporal en proyectos software tiene como objetivo identificar las tareas y asignar el tiempo y recursos necesarios para realizar dichas tareas, de manera que el tiempo de realización del proyecto sea el mínimo posible. La planificación temporal tiene un impacto considerable en el coste final del producto, en su calidad y en la satisfacción final de clientes y usuarios.

Al disponer solamente de una persona en el equipo de trabajo, la planificación de las tareas en los sprints será lineal, por lo que no será necesario tener en cuenta la paralelización de tareas entre distintos miembros del equipo de desarrollo. A la hora de realizar la estimación se ha tenido en cuenta que muchas de las tecnologías que van a ser empleadas en el proyecto son desconocidas para el autor en el momento del inicio de este, por lo que se ha reservado un margen de horas para formación e investigación.

Para la planificación temporal inicial del proyecto, se han tomado como inspiración los conocidos "*The 7 Steps of Machine Learning*" ("Los 7 Pasos del Machine Learning" en español), de Yufeng Guo [4], que se resumen en los siguientes puntos:

1. Adquisición de los datasets o conjunto de datos
2. Preparación de los datos
3. Elección del modelo
4. Entrenamiento del modelo
5. Evaluación del modelo
6. Ajuste de parámetros
7. Predicción

A continuación, se muestra la planificación temporal inicial del proyecto. El proyecto se realizará a lo largo de cuatro sprints:

1. **Adquisición y preprocesado de datasets:** Además de reunir los datasets necesarios para realizar la clasificación y procesarlos, durante este sprint también será necesario generar una gran cantidad de documentación, ya que también incluye un estudio de posibles metodologías a aplicar (de entre las cuales se ha escogido aplicar Scrum), además de un análisis de las tecnologías a emplear. Todo esto unido a la Planificación de tiempo y costes que se desarrolla en esta sección.

2. **Aplicación de baselines:** A lo largo de este sprint se aplicarán una serie de métodos de procesamiento del lenguaje natural para clasificar el dataset, estos métodos se emplearán como baselines para la red neuronal que se implementará en el tercer sprint.
3. **Implementación de la red neuronal recurrente:** Durante este sprint se realizará la implementación de la red neuronal recurrente en TensorFlow.
4. **Ajuste de parámetros y análisis de los resultados:** Ajuste de los hiperparámetros del modelo hasta alcanzar los resultados que se consideren óptimos.

Dentro del número de horas estimado para cada Sprint, se ha tenido también en consideración el tiempo requerido para realizar la documentación correspondiente. Además del tiempo asignado para cada sprint, también deben tenerse en cuenta:

- 40 horas para análisis del problema e investigación
- 20 horas para reuniones
- 20 horas para la preparación de la presentación del proyecto.

Esto hace un total de 300 horas estimadas para la realización del proyecto, tal y como se muestra en la Tabla 1.

Planificación temporal inicial			
Sprints	**Fecha Inicio**	**Fecha Fin**	
Sprint 1: Adquisición y preprocesado de datasets	17/01/2020	05/02/2020	40 horas
Sprint 2: Aplicación de Baselines	06/02/2020	03/03/2020	60 horas
Sprint 3: Implementación de red neuronal recurrente	04/03/2020	10/04/2020	60 horas
Sprint 4: Ajuste de parámetros y análisis de los resultados	11/04/2020	30/04/2020	60 horas
Total Sprints			**220 horas**
Análisis e investigación			**40 horas**
Reuniones			**20 horas**
Presentación			**20 horas**
TOTAL			**300 horas**

Tabla 1: Planificación Temporal Inicial

En la Ilustración 1, se muestra un diagrama de Gantt de la planificación general del proyecto.

Ilustración 1: Diagrama Inicial para la Planificación Inicial del Proyecto

2.2. Planificación de costes

En este apartado se lleva a cabo la Planificación de los costes estimados que deberá asumir el Proyecto para su total consecución. El objetivo principal de la planificación de costes consiste en definir cuál será el presupuesto necesario para completar la ejecución e implementación del proyecto dentro de los límites establecidos en la planificación temporal. Una correcta planificación de costes es un factor determinante a la hora de realizar la venta de un proyecto, ya que una mala planificación con una estimación de presupuesto insuficiente puede suponer una pérdida de dinero para la empresa desarrolladora, mientras que una estimación de presupuesto excesiva puede producir desacuerdos con el cliente.

Se estimarán los costes de los diferentes perfiles de trabajador según las horas desarrolladas bajo cada rol, así como los costes directos e indirectos del proyecto, tales como hardware, software y servicios requeridos.

Cabe destacar que esta planificación es una aproximación inicial que puede sufrir cambios. Por este motivo, se realizarán estudios sobre las desviaciones con respecto a la planificación inicial conforme avance el desarrollo del proyecto.

2.2.1. Costes directos

Los costes directos son aquellos que tienen una relación directa a la realización y producción de productos o servicios ofrecidos por una empresa, como los costes del personal o mano de obra y la materia prima. Estos costes pueden medirse y asignarse de forma directa e inequívoca a un producto concreto [5]. Esta categoría de coste está clasificada en función de su relación con la producción.

2.2.1.1. Coste de personal

A pesar de que el proyecto va a ser realizado por una sola persona, su complejidad es suficiente como para requerir al menos de dos roles claramente diferenciados, estos roles serán el de Jefe de Proyecto y Científico de Datos. El Jefe de Proyecto se encargará de llevar a cabo las labores de documentación y organización, mientras que el Científico de Datos realizará las labores de Investigación e Implementación.

Para realizar el cálculo de los costes del personal se tomará como base el convenio de las TIC del Boletín Oficial del Estado, publicado el 6 de marzo de 2018 [6], que especifica el salario mínimo para cada uno de los puestos mostrado en la Tabla 2. El cálculo del salario medido en euros por hora ha sido realizado dividiendo el sueldo bruto entre el número de horas trabajadas en un período anual.

ROL	Salario	Euros/Hora
Jefe de Proyecto	26.790,31€	26.790,31 / 1800h = 14,88
Científico de Datos	15.860,56€	15.860,56 / 1800h = 8,81

Tabla 2: Costes de Personal por hora

Los salarios mostrados en la Tabla 2 son salarios en bruto, es necesario añadir los costes de Seguridad Social que corren a cargo de la empresa, estos cargos se calculan aplicando porcentajes determinados en el régimen de la Seguridad Social del año 2019 [7]. La base de

cotización se calcula como la suma de todos los conceptos salariales de la nómina del trabajador, que se corresponde con las cifras de la Tabla 2. La Seguridad Social a cargo de la empresa cubre los siguientes riesgos:

- **Contingencias comunes:** Cubre aspectos como enfermedades comúnes, bajas por accidentes no laborales, jubilación y descansos durante el embarazo o el riesgo durante la lactancia natural. Este coste se calcular aplicando un 23,6 % a la base de cotización de cada trabajador.
- **Desempleo:** Cubre el riesgo de quedar sin trabajo en el caso de las personas que se encuentran en posición de querer y poder trabajar. El porcentaje por aplicar varía en función del tipo del contrato que tenga cada trabajador. En este caso, al tratarse de un proyecto temporal, se aplicará el caso de contrato temporal a tiempo parcial, que asciende a un 6,7% por trabajador.
- **FOGASA (Fondo de Garantía Social):** Cubre el riesgo de impago a los trabajadores por causas derivadas de la declaración de insolvencia por parte de los contratadores. Este coste es un 0,2% de la base de cotización de cada trabajador.
- **Riesgo Laboral:** Cubre el riesgo de accidente de trabajo y enfermedades profesionales. Se divide en la cotización por incapacidad temporal (IT), cuyo porcentaje es de 0.65% y, por otra parte, del riesgo de incapacidad permanente, muerte o supervivencia (IMS) derivados del ejercicio de la actividad laboral. Esta cantidad asciende al 1% de la base salarial. La suma de IT e IMS corresponde a un total de un 1,65% de la base salarial.
- **Formación Profesional:** Este apartado se refiere a aportaciones realizadas para que los trabajadores puedan verse beneficiados por cursos de formación profesional para su reciclaje y evolución profesional. Este apartado aplica un 0,6% sobre la base salarial.

El porcentaje total que aplicar sobre la base de cotización de cada trabajador a aportar a la Seguridad Social por parte de la empresa es el mostrado en la Tabla 3:

Cotización	Porcentaje a Aplicar
Contingencias Comunes	23,6%
Desempleo	6,7%
FOGASA	0,2%
Riesgo Laboral	1,65%
Formación Profesional	0.6%
TOTAL	32,75%

Tabla 3: Costes de la Seguridad Social de la Empresa

Una vez aplicado el porcentaje de la seguridad social, el coste por hora de cada perfil es el mostrado en la Tabla 4:

Rol		Euros/Hora
Jefe de Proyecto	14,88€/h + (14,88€ * 32,75/100)	19,75
Científico de Datos	8,81€/h + (8,81€ * 32,75/100)	11,70

Tabla 4: Costes de Personal por hora una vez aplicado el Porcentaje de la Seguridad Social

El cálculo de los costes de personal se calcula por cada tarea, duración de esta y el rol de la persona que lo realiza. En la Tabla 5 se muestra el desglose del coste estimado de la realización de cada tarea:

Actividad	Rol	Salario (euros/hora)	Duración (Horas)	Coste (euros)
Análisis e Investigación				
Análisis e Investigación	Jefe de Proyecto	19,75	0	0
Análisis e Investigación	Científico de Datos	11,70	40	468
		TOTAL ACTIVIDAD	40	468
Reuniones				
Reuniones	Jefe de Proyecto	19,75	20	395
Reuniones	Científico de Datos	11,70	0	0
		TOTAL ACTIVIDAD	20	395
Sprint 1: Adquisición y Preprocesado de Datasets				
Estudio de Metodologías	Jefe de Proyecto	19,75	5	98,75
Planificación	Jefe de Proyecto	19,75	9	177,75
Documentación	Jefe de Proyecto	19,75	10	197,5
Preprocesado	Científico de Datos	11,70	8	93,6
Reunir Conjuntos de Datos	Científico de Datos	11,70	8	93,6
		TOTAL ACTIVIDAD	40	661,2
Sprint 2: Aplicación de Baselines				
Documentación	Jefe de Proyecto	19,75	10	197,5
Documentación	Científico de Datos	11,70	15	175,5
Implementación	Científico de Datos	11,70	35	409,5
		TOTAL ACTIVIDAD	60	782,5
Sprint 3: Implementación de Red Neuronal Recurrente				
Documentación	Jefe de Proyecto	19,75	10	197,5
Documentación	Científico de Datos	11,70	10	117
Implementación	Científico de Datos	11,70	40	468
		TOTAL ACTIVIDAD	60	782,5
Sprint 4: Ajuste de parámetros y análisis de los resultados				
Documentación	Jefe de Proyecto	19,75	20	395
Documentación	Científico de Datos	11,70	10	117
Implementación	Científico de Datos	11,70	30	351
		TOTAL ACTIVIDAD	60	863
Presentación				
Preparación de la Presentación	Jefe de Proyecto	19,75	20	395
Preparación de la Presentación	Científico de Datos	11,70	0	0
		TOTAL ACTIVIDAD	20	395
		TOTAL PROYECTO	**300**	**4347,2**

Tabla 5: Costes de Personal por Tareas

Finalmente, como podemos ver en el estudio realizado en la Tabla 5, el coste de personal total del proyecto asciende a 4347,2 €.

2.2.1.2. Costes del material técnico

2.2.1.2.1. Hardware

Solamente hay que tener en cuenta el equipo informático empleado. Teniendo en cuenta que el período de vida útil de un ordenador personal es de 4 años y que el desarrollo del proyecto tomará aproximadamente cuatro meses, es posible estimar el coste que supondrá el equipo mediante el cálculo de una amortización.

Se ha utilizado un portátil para las tareas realizadas con investigación, ofimática y documentación, con un precio de 530 euros en el momento de comprarlo, mientras que, para las tareas relacionadas con el cálculo y aplicación de los algoritmos, ha sido necesario emplear un ordenador de sobremesa valorado en 1000 euros.

Para estimar el valor de la amortización, se empleará el método de amortización lineal o de cuotas fijas [8], que distribuye de manera equitativa el valor del activo a lo largo de su vida útil. Se calcula aplicando la Ecuación 1.

$$Amortización = \frac{(valor\ de\ reemplazo - valor\ residual)}{vida\ útil}$$

Ecuación 1: Fórmula de la Amortización lineal

De la misma forma, el cálculo de la vida útil en horas se realiza teniendo en cuenta que el equipo solo se usará en días laborables y un determinado número de horas, de acuerdo con la Ecuación 2.

$$\boldsymbol{vida\ útil = nº\ años * 12\ (meses) * 4(semanas) * 5(días\ laborables) * 5(horas\ de\ uso/día)}$$

Ecuación 2: Expresión para calcular la vida útil en horas

Finalmente, el coste del componente del equipo se calcula multiplicando el valor de la amortización en euros por hora por el número de horas estimadas para el proyecto, en este caso 300, tal y como se muestra en la Ecuación 3.

$$coste = Amortización * número\ total\ de\ horas$$

Ecuación 3: Expresión para calcular el coste de un componente hardware

Portátil				
Valor de Reemplazo (€)	Valor Residual (€)	Vida útil (h)	Amortización (€/h)	Coste (€)
530	140	4800	0,08125	24,38

Tabla 6: Coste del portátil

Ordenador Sobremesa				
Valor de Reemplazo (€)	Valor Residual (€)	Vida útil (h)	Amortización (€/h)	Coste (€)
1000	300	4800	0,145833	43,75

Tabla 7: Coste del Ordenador de Sobremesa

De acuerdo con los cálculos mostrados en la Tabla 6 y en la Tabla 7, el coste hardware del proyecto asciende a un total de 68,13 €.

2.2.1.2.2. Software

En lo que respecta a los costes de Software, es necesario adquirir licencias de Windows 10 profesional en los dos equipos disponibles. En el caso del ordenador portátil, el sistema operativo ya venía incluido con la compra de este, sin embargo, para el ordenador sobremesa se ha tenido que adquirir una licencia por un coste de 259 €, este es el precio al que se encontraban las licencias en la página oficial de Microsoft a la hora de redactar el presente presupuesto [9].

También se requiere tener Microsoft Office instalado en ambos dispositivos, necesario para las todas las tareas relativas a la generación de documentación, además del uso de Microsoft PowerPoint para la presentación. Se ha decidido adquirir la licencia Enterprise de Microsoft Office en su versión ProPlus, que es la que resulta más económica al mismo tiempo que cubre todas las necesidades del equipo de trabajo, con un precio de 12,80 € por usuario al mes en el momento de estimar este presupuesto, de acuerdo con la página oficial de Microsoft [10]. El coste de esta licencia será por tanto de 51,20€ por equipo, ya que el desarrollo del proyecto tomará cuatro meses.

El resto de las tecnologías empleadas para el desarrollo del proyecto, que son explicadas con mayor detalle en la sección Tecnologías, son o bien tecnologías de código abierto, y por lo tanto no tienen coste alguno, o se emplean versiones gratuitas. El coste de estos productos asciende a cero euros. En la Tabla 8 se muestra el coste total del proyecto en el apartado Software.

Producto Software	Coste (€)
Licencia Windows 10 Professional	259
Licencia Microsoft Office Sobremesa	51,20
Licencia Microsoft Office Portátil	51,20
TOTAL	361,40

Tabla 8: Coste del Software

De acuerdo con los cálculos realizados en la Tabla 8, el coste del Software del proyecto es de 361,40€.

2.2.1.2.3. Servicios

El único Servicio de pago necesario para la realización del proyecto es el pago de la conexión a internet durante el desarrollo de este, que abarcará cuatro meses. Se ha contratado una conexión de fibra óptica de 600mb simétricos a un precio de 35€ al mes, por lo que el coste de la conexión supondrá un total de 140€.

En la Tabla 9 se desglosa el coste total de materiales del proyecto especificado en las secciones previas, alcanzando un total de 569,13€:

Tipo de Material	Coste (€)
Hardware	68,13
Software	361,40
Servicios	140
TOTAL	569,53

Tabla 9: Coste de Materiales

2.2.2. Costes Indirectos

Los costes indirectos son aquellos que afectan al desarrollo en general y que van asociados a un conjunto de tareas a realizar. Estos costes no pueden ser aplicados a un producto específico, inciden sobre varias actividades de la empresa y no se incorporan físicamente en el producto al finalizarlo, sino que son una parte del proceso. Dentro de este apartado, suelen incluirse costes como las infraestructuras necesarias o la amortización de ciertas herramientas. Para este proyecto no ha sido necesario considerar ninguna infraestructura, ya que el trabajo se ha realizado desde casa, sin necesidad de contratar ningún alquiler. En lo referente a la amortización del equipo, este se ha incluido dentro del apartado Hardware de los costes directos, se ha tomado esta decisión porque se ha tenido en cuenta como un coste material, aunque también podría haberse incluido dentro de esta sección.

Por estas razones, en cuanto a costes indirectos, solo se tendrán en cuenta el impuesto de sociedades y las ganancias y beneficios.

2.2.2.1. Ganancias y beneficios

Suele calcularse definiendo un porcentaje según la política de la empresa para obtener unos beneficios propios para la compañía. Este coste se aplica sobre el presupuesto acumulado para la ejecución del proyecto, el cual es de 4.916,73€. Se ha aplicado una política de un 20% sobre el presupuesto como beneficio propio de la empresa. Por ello, el coste de las ganancias y beneficios es de 983,35€.

2.2.2.2. Impuesto de Sociedades

El impuesto de sociedades [11], o impuesto de la renta, es un impuesto de carácter personal que recae sobre los beneficios obtenidos por la compañía. Este impuesto se calcula aplicándole un 25% al beneficio obtenido por el trabajo desempeñado, por lo que el coste del Impuesto de Sociedades asciende a 245,84€.

2.2.3. Coste total

Una vez realizados los cálculos de los costes directos e indirectos, solo falta sumarlos para obtener el coste total del proyecto, tal y como se muestra en la Tabla 10:

Tipo	Coste (€)
Coste de personal	4347,2
Material técnico	569,53
Ganancias y beneficios	983,35
Impuesto de Sociedades	245,84
TOTAL	6145,92

Tabla 10: Coste Total del Proyecto

En la Ilustración 2, se muestra la distribución de los gastos en forma de gráfico.

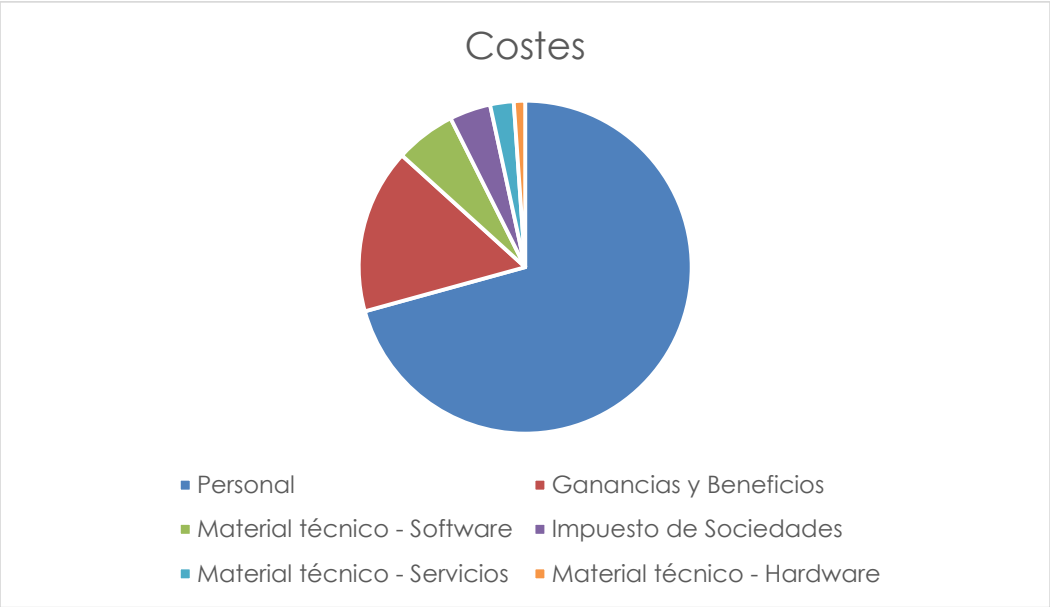


Ilustración 2: Distribución de los gastos

PARTE II. MATERIAS RELACIONADAS

3. Metodología

3.1. Introducción

Uno de los primeros pasos a la hora de planificar el desarrollo de un determinado software es seleccionar la metodología que mejor se adapta al tipo de software y a las condiciones en las que se va a desarrollar con el objetivo de obtener el mejor producto posible [12, 13].

Actualmente existen una gran cantidad de metodologías para abordar el desarrollo de Software, no obstante, es vital conocer la naturaleza del producto a desarrollar antes de elegir una u otra metodología. A causa del constante desarrollo de productos software, la creciente demanda y con el objetivo de facilitar los procesos de desarrollo, se han propuesto una gran cantidad de modelos y metodologías de desarrollo de software. Se identifican principalmente dos enfoques claramente diferenciados: metodologías tradicionales y ágiles.

Las metodologías tradicionales se basan en el uso exhaustivo de documentación durante todo el ciclo de desarrollo del proyecto. Las metodologías ágiles se caracterizan por tener una gran capacidad de respuesta al cambio y por considerar la comunicación con el cliente como un factor de gran prioridad.

Los principales aspectos para tener en cuenta a la hora de seleccionar una metodología u otra son las necesidades de negocio, la percepción del cliente y el tiempo de desarrollo disponible.

3.2. Metodologías tradicionales

Las metodologías tradicionales, también conocidas como metodologías pesadas, se caracterizan por centrarse en la planificación, control y especificación de los requisitos de forma precisa.

Las metodologías tradicionales se basan en organizar claramente las fases del proceso de desarrollo software desde el inicio. El ciclo de desarrollo es unidireccional, con una serie de fases claramente diferenciadas como las mostradas en la Ilustración 3. El desarrollo de documentación exhaustiva es también una de las principales características de esta clase de desarrollo.

Ilustración 3 Ejemplo de Fases de una metodología tradicional

Este tipo de metodología suele emplearse cuando los requisitos pueden definirse con gran precisión desde el principio, esto suele ocurrir cuando se conoce perfectamente como debe ser el producto final y el equipo de desarrollo tiene conocimientos sólidos sobre el dominio del problema y las tecnologías a emplear. Además de esto, es preferible que el dominio del

problema no esté sometido a muchos cambios, lo que podría suponer que una parte importante de los requisitos deba ser revisada, esto sería especialmente perjudicial y costoso durante fases avanzadas del desarrollo.

La principal desventaja de este tipo de metodología es que el usuario no puede llevar a cabo evaluaciones intermedias del producto ni proporcionar feedback sobre si el producto que se está desarrollando satisface sus necesidades, esto hace que el usuario reciba un software basado en el entendimiento del desarrollador sobre el dominio del problema.

Algunas metodologías tradicionales son Metodología en cascada, Método de prototipos, Modelo Incremental, Modelo en Espiral o Métrica V3.

3.3. Metodologías ágiles

Las metodologías ágiles surgen con el objetivo de solucionar algunos de los problemas de las metodologías tradicionales, tales como la falta de capacidad de respuesta al cambio y la escasa comunicación con el cliente.

Estas metodologías permiten la adaptación de la forma de trabajo a las necesidades del proyecto en cada momento, esto permite la redefinición de requisitos y actividades durante el desarrollo del producto.

En febrero de 2001, un grupo de críticos de las metodologías tradicionales de desarrollo se reunieron en Snowbird, Utah con el objetivo de tratar sobre técnicas y procesos relacionados con el desarrollo de Software. En esta reunión se creó la conocida como “The Agile Alliance”, organización sin ánimo de lucro que tiene como objetivo promover los conceptos del Manifiesto Ágil, definido en dicha reunión y en el que se resume la filosofía del enfoque de desarrollo ágil. El Manifiesto ágil se resume en los siguientes cuatro valores: [14, 15]

a) **Individuos e interacciones** sobre procesos y herramientas
   Aunque los procesos ayudan al trabajo y las herramientas mejoran la eficiencia, se requieren personas con conocimiento técnico y una actitud adecuada para obtener resultados. Se considera el recurso humano como el principal factor de éxito.
   Es recomendable comenzar conformando un equipo adecuado y que este defina las herramientas y procesos más apropiados, y no al revés. Unas herramientas y procesos inadecuados pueden obstaculizar al equipo en lugar de actuar como ayuda y soporte para guiar el trabajo.
   Los cambios rápidos que se dan en todos los ámbitos provocan que los procesos y las herramientas de una organización deban cambiar ágilmente, por ello es vital que las personas propongan iniciativas de cambio o se adapten rápidamente al mismo.

b) **Software funcionando** sobre documentación extensiva
   Los documentos actúan como soporte del software, estos permiten la transferencia de conocimiento, registran información histórica y son obligatorios en muchas cuestiones legales o normativas, no obstante, se deben considerar menos importantes que los

productos que funcionan, ya que estos últimos son más trascendentales a la hora de aportar valor al producto.

La comunicación entre personas y la interacción con prototipos pueden aportar mucho más valor que una documentación exhaustiva. Se llega por lo tanto a la conclusión de que el uso de la documentación debe reducirse al mínimo indispensable, ya que genera trabajo que no aporta un valor directo al producto.

Difícilmente se puede conseguir que un documento contenga requisitos detallados antes de comenzar un proyecto.

c) **Colaboración con el cliente** sobre negociación contractual

Se espera que el software sea capaz de satisfacer las necesidades exigidas o esperadas por el cliente, dentro del plazo establecido.

En las Metodologías Tradicionales, es frecuente que el cliente y el equipo de desarrollo asuman posiciones distantes y que dediquen una gran cantidad de tiempo a la tarea de redactar, depurar y firmar el contrato.

Por el contrario, las Metodologías ágiles pretenden involucrar de manera directa y comprometida al cliente o usuario final dentro del equipo de trabajo. La participación del cliente y su interacción con el equipo de desarrollo deben ser constantes durante todo el desarrollo del proyecto.

El cliente es el que mejor sabe que es lo que necesita o desea, por lo que es el más adecuado para hacer recomendaciones o correcciones en cualquier momento del proyecto, y esto solo se puede conseguir con una implicación adecuada y constante.

d) **Respuesta al cambio** sobre seguir un plan

La naturaleza cambiante de la tecnología y la dinámica de la sociedad moderna provocan que un proyecto de desarrollo software deba enfrentarse con frecuencia a cambios durante su ejecución.

Estos cambios pueden deberse a factores tan variados como ajustes en la personalización del software, cambios en las leyes, comportamiento de la competencia, nuevas tendencias tecnológicas, entre otros. Las metodologías tradicionales pretenden tener todo completamente definido desde el comienzo, esto resulta poco realista dada la complejidad general de un proyecto software.

La capacidad de respuesta a cambios imprevistos tiene más valor que la capacidad de seguir un plan de seguimiento preestablecido. Los principales valores de la gestión ágil son la anticipación y la adaptación al cambio, esto resulta más efectivo que realizar una planificación y un control exhaustivos para evitar desviaciones sobre el plan inicial.

3.4. Metodologías tradicionales vs Metodologías ágiles

Ilustración 4 Comparación de la estructura de una Metodología ágil con una tradicional

Las metodologías tradicionales son idóneas en proyectos en los que se afronta un problema conocido y una solución bien definida y acotada [16, 17]. En esta clase de situaciones resulta más sencillo analizar, diseñar y ejecutar una solución. Suelen emplearse principalmente cuando se sabe que los requisitos establecidos al principio del desarrollo no van a cambiar, esto es idóneo en proyectos en los que el riesgo es más limitado o el equipo de desarrollo conoce el dominio del problema y está familiarizado con las tecnologías que se van a emplear.

Por el contrario, si el entorno es cambiante y la forma de solucionar el problema planteado no está del todo clara, lo más adecuado es emplear una metodología ágil, esto permitirá al equipo de desarrollo adaptarse al cambio con mayor facilidad.

En el caso de este proyecto, al momento de iniciarlo aún no se tienen los conocimientos necesarios de procesamiento de lenguaje natural (dominio del problema) ni de las tecnologías a emplear, esto provoca que no sea viable emplear una metodología tradicional, ya que este proyecto será realizado al mismo tiempo que se aprenden nuevas tecnologías y se adquieren nuevos conocimientos, por lo que es de esperar que se sucedan una serie de cambios inesperados que causen bastantes variaciones sobre la planificación inicial. Además de esto, al inicio del desarrollo, muchos de los problemas que pueden llegar a aparecer son desconocidos. Por estas razones, se ha llegado a la conclusión de que la metodología más apropiada para afrontar el problema a desarrollar debe ser una metodología ágil.

3.5. Estudio de metodologías ágiles

Con el objetivo de determinar cuál es la metodología ágil que mejor se adapta al proyecto a desarrollar, a continuación, se realiza un pequeño estudio comparativo de algunas de las metodologías ágiles más populares.

3.5.1. XP

La metodología XP, también conocida como eXtreme Programming o programación extrema [18] es una metodología ágil que da prioridad a la adaptabilidad frente a la previsibilidad, ya que los defensores de esta metodología sostienen que los cambios en los requisitos durante el desarrollo son un aspecto natural e inevitable del mismo, en muchos casos incluso deseable.

Los cinco valores que define la metodología XP son la comunicación como medio de transmisión de conocimiento entre los miembros del equipo, la simplicidad tanto del código como de la documentación, la retroalimentación (o feedback) constante por parte del cliente, la valentía a la hora de saber adaptarse a los cambios y ser persistente al resolver un problema, y por último el respeto entre los miembros del equipo.

Las características fundamentales de este método son el desarrollo iterativo e incremental, las pruebas unitarias continuas, la programación en parejas, la integración del equipo de programación con el cliente, la corrección de todos los errores antes de añadir una nueva funcionalidad (con entregas frecuentes), refactorización y simplicidad del código y la compartición de la propiedad del código entre todo el equipo.

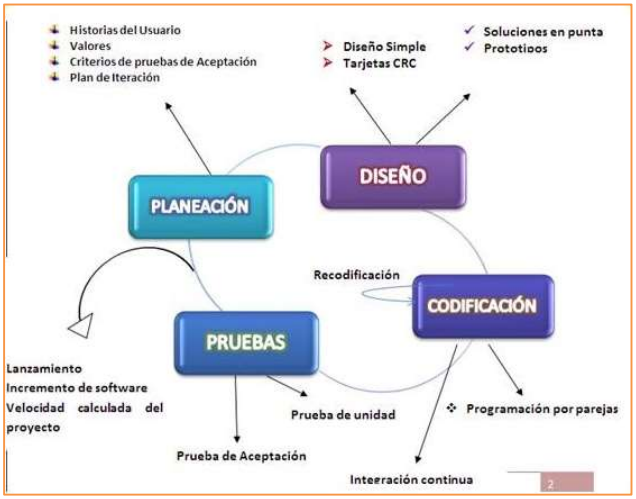


Ilustración 5 Esquema del ciclo de desarrollo de un proyecto XP

En la Ilustración 5 se muestra el esquema del ciclo de desarrollo de un proyecto XP [19]. Durante la fase de Planificación se reúnen los requisitos que permitirán a los miembros técnicos del equipo XP entender el contexto del negocio para el software. Esto lleva a la creación de historias de usuario para las que el cliente asigna un valor y el equipo le asigna un tiempo estimado, en caso de que el tiempo estimado para una historia de usuario sea superior a tres semanas, se le pide al cliente que descomponga dicha historia en varias historias más simples. En futuras entregas se evalúa si se ha cumplido con las estimaciones realizadas durante la fase de planificación anterior y, en función de las conclusiones a las que se lleguen, se determina si es necesario modificar el contenido de las entregas o cambiar las fechas de entrega.

Durante la fase de Diseño, se prefiere un diseño sencillo sobre una representación compleja. El único producto que se genera a lo largo de esta fase son las tarjetas CRC (clase-responsabilidad-colaborador) que identifican las clases orientadas a objetos que son relevantes para el incremento actual del software. Esta fase también se caracteriza por la creación de prototipos del diseño para evaluarlos con el objetivo de minimizar el riesgo cuando comience la implementación, validando de esta forma las estimaciones originales, esto se conoce como solución en punta.

La fase de Codificación comienza realizando las pruebas unitarias de cada una de las historias de la entrega en curso, de esta forma, a la hora de realizar la implementación, el desarrollador ya está capacitado para centrarse en lo que se debe implementar. Una vez que el código está terminado, se aplican las pruebas unitarias diseñadas previamente, por lo que se obtiene una retroalimentación instantánea para los desarrolladores. A medida que los programadores terminan su trabajo, el código desarrollado se integra con el trabajo realizado por el resto de los programadores, esto se conoce como integración continua.

Las pruebas unitarias realizadas durante la fase de codificación deben implementarse de manera que puedan ser automatizadas, esto facilita la detección de errores durante la integración de componentes o cuando se realizan cambios inesperados. Las pruebas de aceptación XP, también conocidas como pruebas del cliente se centran en la funcionalidad general del sistema, la parte que es visible por el cliente. Suelen derivar de las historias de usuario.

3.5.2. Kanban

Esta metodología ágil es estadísticamente la que menos resistencia presenta en las compañías que están acostumbradas a las metodologías tradicionales. La principal ventaja de esta metodología es que es muy fácil de utilizar, actualizar y asumir por parte del equipo. La aplicación de esta metodología implica la generación de un tablero de tareas que permitirá mejorar el flujo de trabajo y alcanzar un ritmo sostenible [20, 21]. Dicho tablero será visible y accesible para todos los miembros del equipo, también deberá tener tantas columnas como estados por los que pasa una tarea, desde que esta inicia hasta que finaliza.

El objetivo de la visualización de tareas mediante un tablero es clarificar al máximo el trabajo a realizar, las tareas asignadas a cada miembro del equipo de trabajo, además de las prioridades y la meta asignada.

Los miembros del equipo deben mantener un flujo de trabajo constante, no pueden empezar una nueva tarea hasta que terminen la que estén realizando en ese momento, de manera que solo se realiza trabajo cuando existe capacidad para procesarlo.

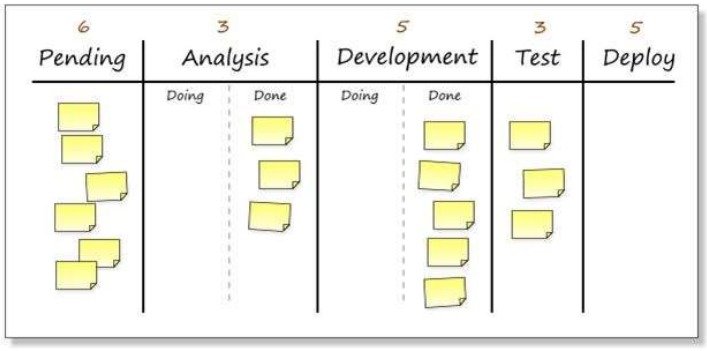


Ilustración 6 Ejemplo de un tablero Kanban

3.5.3. FDD

La metodología FDD, también conocida como Feature-Driven Development es un proceso software pragmático centrado en el cliente y la arquitectura [22]. Se caracteriza por llevar a cabo

solo las fases de diseño e implementación en un proyecto, suele estar enfocado a proyectos muy grandes y críticos.

FDD está compuesto por cinco fases de las cuales las tres primeras son secuenciales y las dos últimas son iterativas:

- **Fase 1:** Desarrollo de un modelo general.
- **Fase 2:** Creación de una lista de requisitos.
- **Fase 3:** Planificación de los requisitos.
- **Fase 4:** Diseño de los requisitos.
- **Fase 5:** Implementación de los requisitos.

Hay seis roles principales en un proyecto de estas características: Administrador del proyecto, Arquitecto jefe, Gestor del desarrollo, Programador Jefe, Propietario de clases y Experto en el Dominio. Un miembro del equipo puede ejercer uno o más roles.

Esta metodología tiene las ventajas de poder adaptarse a los cambios en los requisitos durante todo el desarrollo, permitir entregas continuas y de plazos cortos, además de centrarse en la excelencia técnica y el buen diseño.

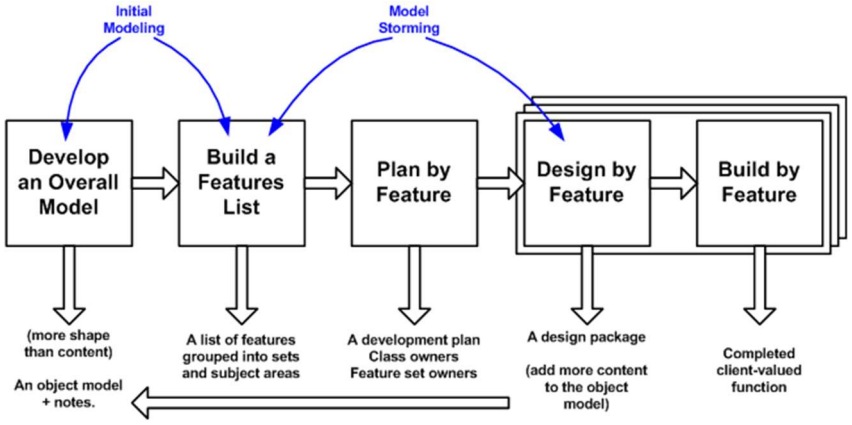


Ilustración 7 Esquema del ciclo de vida FDD

3.5.4. ASD

El método ágil ASD (Adaptative Software Development) es un modelo de implementación de patrones ágiles para el desarrollo de software [23]. Al igual que otras metodologías de desarrollo software, su funcionamiento es cíclico y reconoce que en cada iteración se producirán cambios e incluso errores. Esta metodología se adapta al cambio en lugar de luchar contra él, se basa en la adaptación continua a circunstancias cambiantes. En ella no hay un ciclo de planificación-diseño-construcción del software, sino un ciclo especular-colaborar-aprender.

El desarrollo de software adaptable se caracteriza por ser una metodología:

- Iterativa

- Orientada a los componentes de software más que a las tareas en las que se va a alcanzar dicho objetivo.
- Tolerante a los cambios.
- Guiada por los riesgos.
- En la que la revisión de los componentes sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo.

La Ilustración 8 es un esquema en el cual se indican las distintas fases de un ciclo de vida ASD. Durante la fase de Especulación se establecen los principales objetivos y metas del proyecto, realizando estimaciones de tiempo. También se decide el número de iteraciones necesarias para realizar el proyecto.

La mayor parte del desarrollo se centra en la fase de colaboración, durante esta fase es importante la coordinación entre los miembros del equipo, asegurándose siempre de que lo aprendido se transmita al resto del equipo para que otros miembros no tengan que volver a pasar por el mismo proceso de aprendizaje, con el tiempo que esto conlleva.

La etapa de aprendizaje consiste en recopilar lo que se ha aprendido durante la fase de colaboración, debe recopilarse tanto lo positivo como lo negativo. Es un elemento crítico para la eficacia de los equipos. Durante esta etapa, hay cuatro tipos de aprendizaje:

- Calidad del producto desde un punto de vista del cliente.
- Calidad del producto desde un punto de vista de los desarrolladores.
- Gestión del rendimiento del equipo.
- Análisis de la situación del proyecto.

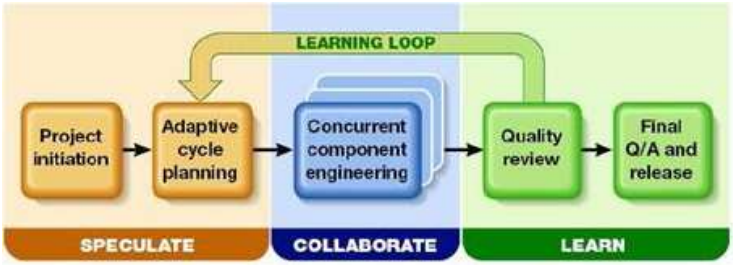


Ilustración 8 Ciclo de vida ASD

Como ventajas de este método podemos destacar la capacidad de aprender de los errores de entregas anteriores e iniciar de nuevo el ciclo de vida, además de que los cambios sirven para mejorar el software y promueven la colaboración e integración de los participantes del proyecto.

El principal inconveniente de este método es que los errores de una iteración suelen corregirse en iteraciones futuras, esto puede provocar que no se detecten a tiempo, afectando de esta forma a la calidad y coste del producto, especialmente si una iteración depende de la anterior.

Como conclusión, este método parece apropiado para ser aplicado en proyectos pequeños o medianos.

3.5.5. SCRUM

3.5.5.1. Introducción

Scrum es una metodología de desarrollo ágil que se caracteriza por:

- Adoptar una estrategia de desarrollo incremental en lugar de una planificación y ejecución completa del producto.
- Basar la calidad de los resultados en el conocimiento de los miembros de un equipo autoorganizado, en lugar de en la calidad de los procesos empleados.
- Solapar las fases del desarrollo en lugar de realizarlas de forma secuencial.

Este modelo fue identificado y definido por los japoneses Ikujiro Nonaka y Hirotaka Takeuchi en el año 1986 en el artículo "The New Product Development Game", que dio origen a una nueva forma de gestionar proyectos que toma la agilidad, flexibilidad y la incertidumbre como elementos principales.

Nonaka y Takeuchi crearon esta metodología basándose en empresas tecnológicas de renombre (Honda, HP, Canon, entre otras) que, trabajando en el mismo entorno que otras empresas, conseguían realizar productos de buena calidad en menos tiempo y con menor coste que estas.

En lugar de emplear un equipo especializado en cada una de las fases de sus proyectos, estas empresas partían de unos requisitos muy generales y empleaban a un equipo multidisciplinar que trabajaba en el producto desde el comienzo del proyecto hasta el final.

La metodología Scrum es adecuada para aquellas empresas en las que el desarrollo de productos se realiza en entornos que se caracterizan por tener:

- **Incertidumbre:** Los objetivos se plantean sin proporcionar un plan detallado del producto. Los puntos clave para llegar a conseguir la incertidumbre en un proyecto se basan en la tensión que supone no tener una idea clara del proyecto combinada con un margen de autonomía, libertad y responsabilidad.
- **Auto-organización:** Los equipos deben tener la capacidad de organizarse por sí solos, no necesitan roles para la gestión, pero tienen que poseer las siguientes características:
    - ***Autonomía:*** Deben ser capaces de encontrar la solución a los problemas planteados empleando para ello la estrategia que consideren más adecuada.
    - ***Auto-superación:*** Capacidad de mejorar las soluciones iniciales.
    - ***Auto-enriquecimiento:*** Los equipos multidisciplinares deben ser capaces de enriquecerse de forma mutua, proporcionando soluciones que puedan complementarse
- **Control moderado:** Establecer un control suficiente como para evitar descontroles. Se sigue un principio de "autocontrol entre iguales" para no impedir la creatividad y espontaneidad de los miembros del equipo.
- **Transmisión del conocimiento:** Como se ha mencionado previamente, los miembros del equipo son multidisciplinares, de manera que todos los miembros aportan y aprenden tanto de sus propias investigaciones como de la experiencia de desarrollo propia y del resto del equipo. La información adquirida y generada por un miembro del equipo debe ser accesible para el resto de los integrantes de este, así como las herramientas y

políticas de gestión del conocimiento. Esta retroalimentación fomenta un clima de trabajo en equipo y evita que se repitan los mismos errores constantemente.

Scrum se caracteriza por el empleado de ciclos breves de desarrollo conocidos como “sprints”. Para entender el ciclo de desarrollo de Scrum es necesario estar familiarizado con las cinco fases que definen el ciclo de desarrollo ágil (Ilustración 9).

Ilustración 9 Fases de un ciclo de desarrollo en Scrum

1. **Concepto:** Durante esta fase, se definen de manera general las características del producto y se asigna al equipo que se encargará del desarrollo de este.
2. **Especulación:** Esta fase consiste en realizar disposiciones con la información obtenida, estableciendo los límites que marcarán el desarrollo del producto, como los costes y la gestión del tiempo. El producto se construirá partiendo de las ideas principales, comprobando siempre las partes realizadas y su impacto en el entorno. Esta fase se repetirá en cada iteración y consiste en:
    - Desarrollar y revisar los requisitos generales.
    - Mantener la lista de funcionalidades esperadas.
    - Realizar un plan de entrega, estableciendo las fechas de las revisiones, hitos e iteraciones.
    - Medir el esfuerzo realizado en el proyecto.
3. **Exploración:** Incrementar el producto mediante la implementación de las funcionalidades acordadas durante la fase de especulación.
4. **Revisión:** Revisar el trabajo realizado y contrastarlo con el objetivo a alcanzar.
5. **Cierre:** Entregar una versión del producto deseado dentro del plazo establecido. No se trata de una versión finalizada de un proyecto, sino de una versión que será sometida a revisiones y a cambios, conocidos como mantenimiento, que hará que el producto se acerque al producto final deseado.

3.5.5.2. Componentes de Scrum

3.5.5.2.1. Roles

En Scrum, los roles se dividen en dos grandes subgrupos, conocidos como cerdos y gallinas, los cerdos son miembros comprometidos con el proyecto mientras que las gallinas son integrantes involucrados en el proyecto. Esta idea proviene del chiste inspirado en el cerdo y la gallina que

se muestra en la Ilustración 10. Los roles “cerdos” se definen como aquellos que lo dan todo de sí por el producto, suponiendo un riesgo para ellos (el equipo de trabajo), mientras que los roles “gallinas” son aquellos interesados en el producto, pero que no perderían nada de sí mismos en el caso de que el proyecto fracasara (clientes). Las necesidades, deseos, ideas e influencias de los roles gallina se tienen en cuenta, pero no de forma que pueda afectar, distorsionar o entorpecer el proyecto Scrum [24].

Ilustración 10 Chiste “Cerdos y gallinas”

3.5.5.2.1.1. Roles Cerdo

- **Scrum Máster:** Se encarga de comprobar que el modelo y la metodología se están aplicando correctamente. Es responsable de eliminar todos los inconvenientes que hagan que el proceso no fluya correctamente e interactuará con el cliente y con los gestores.
- **Product Owner:** Persona responsable de gestionar el Product Backlog y asegurar el valor del trabajo que realiza el equipo. Se encarga de mantener el Product Backlog y asegurar la visibilidad de este para todos los miembros del equipo. Actúa como representante del cliente y se encarga de decidir cómo será el resultado final del producto.
- **Scrum Team:** Suele tratarse de un equipo de entre cinco y nueve personas con autoridad para organizar y tomar decisiones para conseguir su objetivo. Está involucrado en la estimación del esfuerzo de las tareas del Backlog. Se trata de un equipo multifuncional, todos los miembros trabajan de forma solidaria y con responsabilidad compartida.

3.5.5.2.1.2. Roles gallina

Estos miembros comparten la característica de aportar información de gran utilidad. El proyecto puede ser gestionado e implementado sin ellos, pero son necesarios para su completo y correcto desarrollo.

- **Usuarios finales:** Son los usuarios que utilizarán el producto final. No siempre son los propios clientes.
- **Stakeholders** o interesados: Grupo de personas que hacen posible el cumplimiento del proyecto además de las personas que se verán beneficiadas por el mismo.
- **Managers:** Personas encargadas de tomar decisiones finales a la hora de especificar los objetivos y requisitos del proyecto.

3.5.5.2.2. Reuniones

- **Sprint Planning:** Es la tarea de planificación del sprint. Durante esta actividad, los miembros del equipo se reúnen y deciden los requisitos y tareas que se asignarán a cada miembro del equipo, además de estimar el tiempo requerido para cada tarea. De esta forma se establece la duración total del sprint.
- **Sprint Daily:** Se realizarán reuniones diarias en las que se contestarán las siguientes tres preguntas principales para evaluar el avance del proyecto:
    - *¿Qué trabajo se realizó desde la reunión anterior?*
    - *¿Qué trabajo se realizará hasta la próxima reunión?*
    - *Inconvenientes que han surgido y que deben resolverse para poder continuar*

  Estas reuniones tienen como finalidad poner en común y sincronizar las actividades para elaborar un plan del día. Además, si un integrante tiene algún inconveniente, debe darlo a conocer para buscar propuestas de solución en equipo.
- **Backlog Refinement:** El Product Owner revisa el estado de las actividades a implementar indicadas en el Product Backlog, con el fin de aclarar cualquier duda que pueda surgir por parte de los desarrolladores. Esta reunión también sirve para realizar estimaciones sobre el tiempo y el esfuerzo realizado por cada integrante del equipo, además de tareas de planificación de cara al siguiente sprint.
- **Sprint Review:** Una vez finalizado el Sprint se realizará una revisión del incremento generado. Los resultados finales se presentarán, acompañados de una demo o versión, esto ayuda a mejorar el feedback por parte del cliente.
- **Sprint Retrospective:** Esta reunión se realiza con el objetivo de mejorar el proceso de trabajo y aplicar cambios propuestos de cara a futuros sprints. Durante la reunión se plantean las siguientes preguntas para cada miembro del equipo:
    - *¿Qué se ha hecho bien durante el sprint?*
    - *¿Qué se ha hecho mal durante el sprint?*
    - *¿Qué problemas han surgido durante el sprint?*

3.5.5.2.3. Elementos de Scrum

- **Product Backlog:** Listado de tareas que se pretenden realizar durante del desarrollo del proyecto. Este listado debe ser visible para todo el equipo y debe disponer de una visión amplia y general sobre lo que se espera realizar.
  Este listado suele estar compuesto de diferentes tipos de elementos o tareas a realizar ordenados por prioridad.
- **Sprint Backlog:** Se trata de un subconjunto de tareas extraídas del Product Backlog con la intención de ser llevadas a cabo durante el sprint actual. Se genera durante el Sprint Planning meeting y está formado por las mismas tareas que serán mostradas al cliente durante el Sprint Review al final de la iteración.
- **Sprint Burndown:** Es un gráfico ilustrativo en el que se representa el avance de las tareas realizadas cada día durante un Sprint. Esta gráfica permite realizar un análisis al final de cada Sprint durante el Sprint Retrospective para identificar días clave en los que el rendimiento del proyecto se vio beneficiado o afectado, además de analizar las posibles desviaciones surgidas con respecto a la planificación inicial [25].

En un gráfico Burndown (Ilustración 11) aparecen dos series temporales: La serie ideal y la real. El objetivo es realizar un flujo de trabajo que se asemeje lo máximo posible a la serie ideal.

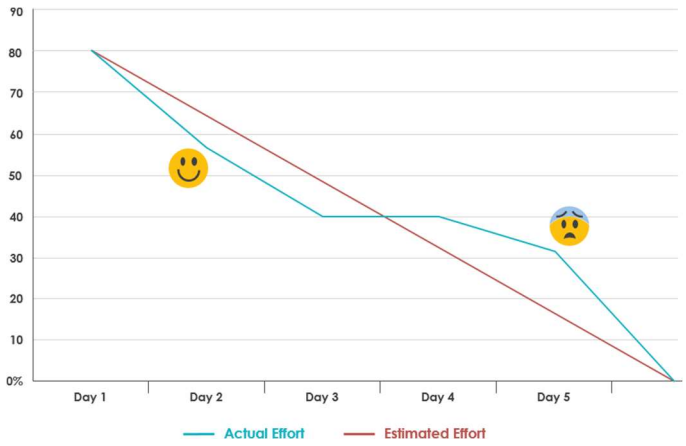


Ilustración 11 Ejemplo de gráfica Sprint Burndown

En conclusión, podemos decir que esta metodología está indicada para proyectos en los que los requisitos estén poco definidos o puedan cambiar con facilidad. No obstante, es importante equilibrar las tareas asignadas a cada Sprint, de manera que estén bien balanceados y las entregas se puedan realizar a tiempo.

3.5.6. Metodología seleccionada: Scrum

Tras estudiar diversas metodologías, se ha llegado a la conclusión de que Scrum es la más adecuada para realizar este proyecto. Esta decisión se ha tomado porque, al tratarse de un proyecto en el que se aprende al mismo tiempo que se realiza el desarrollo, resulta más adecuado seguir un enfoque como el de Scrum, en el que la Incertidumbre es una de sus principales características. Asimismo, resulta más sencillo controlar el flujo de trabajo realizado empleando sprints, ya que al observar el Sprint Burndown, se puede determinar si el proyecto avanza al ritmo deseado.

4. Tecnologías

4.1. Taiga

Taiga [26] es una herramienta de Software libre diseñada para la gestión de proyectos en los que se empleen metodologías ágiles, como Scrum y Kanban. Esta herramienta será utilizada en este proyecto para llevar a cabo el seguimiento de las tareas asignadas a cada sprint.

En la Ilustración 12 se muestra el panel principal de Taiga en un punto dado del desarrollo, con las historias de usuario ubicadas en el centro y los Sprints en la derecha.

Ilustración 12: Panel Principal de Taiga

4.2. Toggl

Toggl [27] es una herramienta de gestión de tiempo que se emplea para cronometrar el número de horas dedicadas a la realización de cada una de las tareas que componen el proyecto.

También permite revisar estadísticas y reportes del tiempo registrado, lo que resultará muy útil para llevar a cabo un análisis constante del rendimiento y la gestión de las tareas. Estos informes serán de especial utilidad para calcular las desviaciones de tiempo y costes una vez finalizado el proyecto.

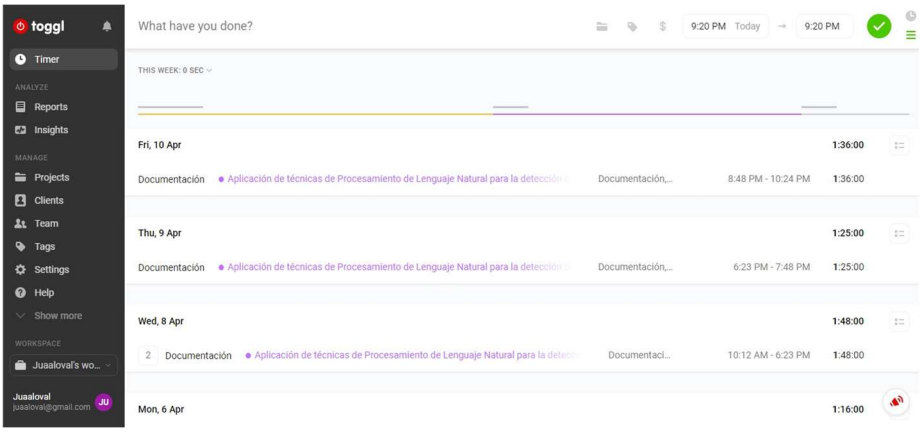


Ilustración 13: Interfaz de Toggl

4.3. PyCharm

PyCharm [28] es un entorno de desarrollo integrado (IDE) diseñado por la compañía JetBrains específicamente para el lenguaje de programación Python. Posee herramientas de análisis y depuración de código, además de integración con sistemas de control de versiones.

Para el desarrollo de este proyecto se utilizará la Community Edition de PyCharm, que ha sido desarrollada bajo los términos de la licencia Apache [29].

4.4. Jupyter Notebook

Los cuadernos de Jupyter [30] (conocidos comúnmente como "Jupyter Notebooks" o simplemente "notebooks") son archivos producidos por la aplicación Jupyter Notebook App, que contienen tanto elementos de texto enriquecido (escritos en notación markdown) como fragmentos de código ejecutable en Python. Este formato permite que los notebooks resulten más legibles para un humano, al contener gráficos y descripciones, esto es especialmente útil al realizar tareas de análisis.

Esta tecnología se empleará en las tareas de análisis del conjunto de datos y en las pruebas, al ser menos exigentes computacionalmente que los modelos neuronales.

4.5. Google Colab

Google Colab es un entorno gratuito de Jupyter Notebook desarrollado por Google que no requiere configuración previa y que se ejecuta en la nube. La versión gratuita nos permite acceder a un equipo de 12 GB de RAM de forma remota, dándonos la opción de ampliar hasta 25 GB sin ningún coste en caso de que sea insuficiente.

Los cuadernos de Colab son cuadernos de Jupyter alojados en la nube, por lo que disponen de todas las características de estos, pero en un entorno más potente, que no requiere de configuración y que permite compartir proyectos con mayor facilidad.

La principal ventaja de esta herramienta es que libera a nuestro equipo de tener que llevar a cabo un trabajo demasiado costoso tanto en tiempo como en potencia o incluso nos permite realizar ese trabajo si nuestra máquina no cuenta con recursos suficientes.

4.6. GitHub

GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git y el Framework Ruby on Rails. Algunas características de GitHub son:

- Seguimiento de problemas entre distintos miembros del equipo.
- Sistema de revisión de código en el que se pueden añadir anotaciones a los ficheros alojados.
- Sistema de visión de ramas entre las distintas versiones.

4.7. Python

Python es un lenguaje de programación cuya filosofía hace hincapié en la legibilidad de su código. Es un lenguaje multiparadigma, ya que soporta orientación a objetos, programación imperativa y programación funcional.

Python es, junto con R, uno de los lenguajes más populares en el ámbito del Machine Learning. Dispone de un amplio catálogo de librerías orientadas a Inteligencia Artificial y Machine Learning, lo que permite reducir considerablemente el tiempo y la dificultad de los desarrollos mediante el empleo de estos frameworks y librerías. Gracias al uso de estas tecnologías, no será necesario reinventar la rueda para implementar características y métodos que ya han sido implementados y optimizados en el pasado.

A continuación, se describen brevemente las principales librerías de Python que van a emplearse durante el desarrollo de este proyecto.

4.7.1. TensorFlow 2.0

TensorFlow [31] es una biblioteca de código abierto desarrollada por Google y que ha alcanzado una gran popularidad en el campo del aprendizaje automático desde su creación. TensorFlow también se caracteriza por tener un tiempo de compilación más rápido que otras bibliotecas de aprendizaje profundo.

Esta librería trabaja con datos en forma de arrays multidimensionales llamados tensores, lo que le permite manejar grandes cantidades de datos. En octubre de 2019 se lanzó la versión 2.0 de TensorFlow, incorporando una gran cantidad de novedades, siendo las principales:

- Facilitar la construcción de modelos empleando keras
- Facilitar el despliegue de los modelos en una gran cantidad de plataformas
- Simplificación general de la API

Entre las principales librerías de aprendizaje automático (TensorFlow 2.0, PyTorch, Keras y FastAI), TensorFlow 2.0 tiene el mayor aumento en las listas de empleos de las principales bolsas de trabajo en línea, con empresas que muestran aproximadamente un 50% más de demanda en esta tecnología, comparada con Keras y por delante de PyTorch para liderar el conocimiento tecnológico (Ilustración 14).

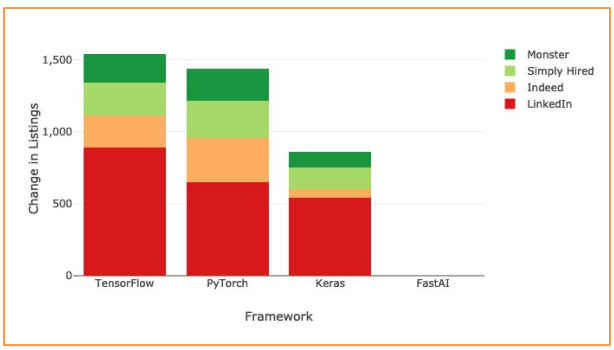


Ilustración 14: Crecimiento de la demanda laboral por Framework [32]

4.7.2. TensorBoard

TensorBoard es un Software de visualización que viene incluido dentro de cualquier instalación estándar de TensorFlow. Fue creado para facilitar el análisis del flujo de información entre tensores, favoreciendo así la depuración y optimización de los modelos. Entre sus principales funciones se encuentran:

- Monitorizar y visualizar métricas como la función de pérdidas y la precisión.
- Visualizar el grafo de computación del modelo.
- Analizar los histogramas de los pesos, sesgos y otros tensores, además de las alteraciones que sufren a lo largo de las épocas de entrenamiento.

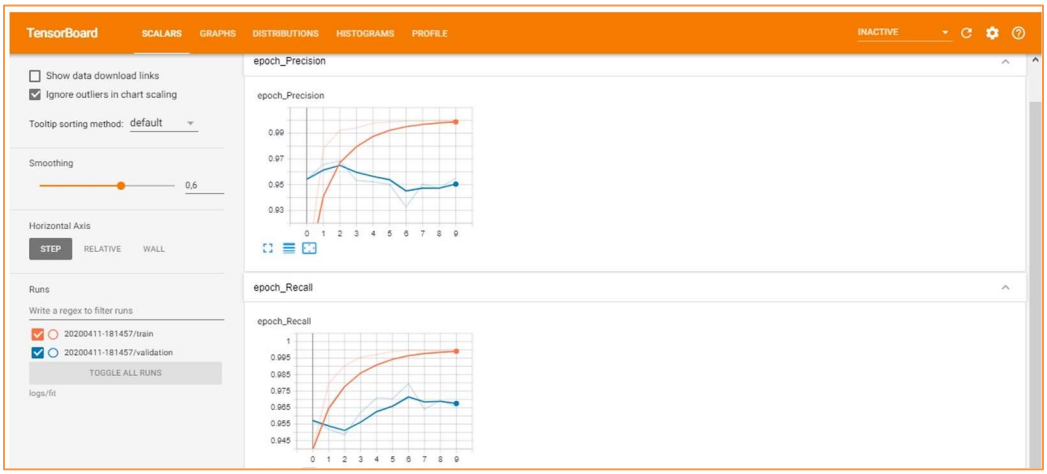


Ilustración 15: Captura de una ventana de TensorBoard

4.7.3. Pandas

En computación y Data Science, pandas [33] es una biblioteca de software escrita como extensión de Numpy para manipulación y análisis de datos. Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales. Algunas de sus características son:

- Permite la manipulación de datos con indexación integrada.
- Contiene herramientas para leer y escribir datos entre estructuras de dato en memoria y formatos de archivo variados.
- Reestructuración y segmentación de conjuntos de datos.
- Inserción y eliminación de columnas en estructuras de datos.
- Mezcla y unión de datos.

La biblioteca ha sido altamente optimizada en cuanto a rendimiento.

4.7.4. Numpy

Numpy [34] es una librería de Python que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con dichos vectores o matrices.

4.7.5. Matplotlib

Matplotlib es una librería que permite crear visualizaciones estáticas, animadas e interactivas en Python.

4.7.6. Nltk

Natural Language Toolkit (nltk) [35] es un conjunto de librerías y programas utilizados para el procesamiento del lenguaje natural en Python. Esta librería se utilizará para realizar el preprocesamiento de los datos.

4.7.7. Sklearn

Scikit-learn [36] es una librería que incluye varios algoritmos de clasificación, regresión y clustering entre los cuales se encuentran Naive Bayes y los modelos basados en Bag of Words.

4.7.8. Gensim

Gensim [37] es una librería diseñada para realizar topic-modeling y procesamiento del lenguaje natural. Utilizaremos esta librería para obtener los Words Embeddings en Word2Vec y Doc2Vec.

4.8. Redes neuronales

Las redes neuronales son un modelo computacional vagamente inspirado en el funcionamiento del cerebro humano. Consiste en un conjunto de unidades, denominadas neuronas artificiales, conectadas entre sí para transmitirse señales. A pesar de tratarse de un modelo muy prometedor, inicialmente no llegó a realizarse en la práctica debido a dos factores principales:

- Ausencia de conjuntos de datos lo suficientemente grandes. Ya que, a pesar de ofrecer muy buenos resultados, las redes neuronales requieren generalmente de una mayor cantidad de datos que otros modelos más simples.
- Necesidad de una mayor potencia computacional. Durante el siglo pasado, los ordenadores eran incapaces de llevar a cabo la ejecución de un modelo tan complejo y, si lo eran, requerían de una gran cantidad de tiempo y recursos, lo que provocaba que otros métodos más simples resultasen más atractivos.

No obstante, en recientes años se ha producido un aumento considerable en la capacidad de computación de los ordenadores gracias a una gran cantidad de avances tecnológicos, hasta el punto de que muchos ordenadores personales de gama media y alta son capaces de ejecutar estos modelos de la manera en la que fueron concebidos.

Unido a esto, gracias al progreso de internet y las nuevas tecnologías, hoy en día podemos tener a nuestra disposición una gran cantidad de datos con los que entrenar nuestros modelos neuronales. Existen incluso algunos portales web como Kaggle [38], cuyos usuarios ofrecen datasets a disposición de todo aquel que quiera descargarlos, permitiendo también compartir diversas soluciones y usos de los datos por parte de la comunidad. Además de esto, siempre está la opción de crear nuestros propios conjuntos de datos empleando técnicas como Web Scraping [39].

Gracias a estos avances, los dos grandes obstáculos que impedían que las redes neuronales fuesen viables en la práctica han sido eliminados, consecuencia de esto ha sido el gran incremento de popularidad de estos modelos en los últimos años, como puede verse en la Ilustración 16, que representa el número de entradas sobre redes neuronales y deep learning en Google Scholar desde 1990 [40].

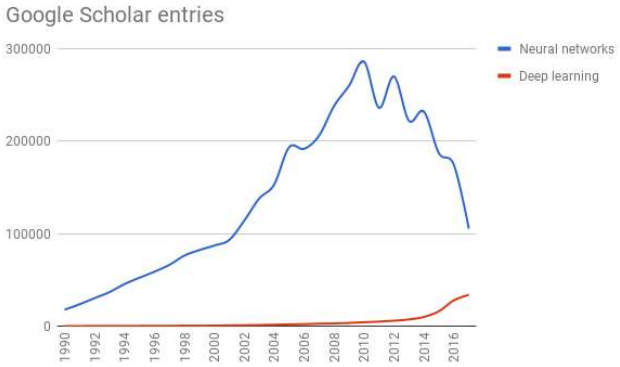


Ilustración 16: Número de entradas referentes a redes neuronales y Deep learning desde 1990

4.8.1. Funcionamiento de las redes neuronales

Las redes neuronales son un modelo matemático basado en una estructura de grafo dirigido cuyos nodos son neuronas artificiales [41]. Estos nodos se conectan entre si mediante aristas que tienen un peso asociado, y se organizan en capas formadas por un gran número de ellos. Las redes neuronales compuestas por más de una capa se conocen como redes neuronales multicapa (la mayoría de las redes neuronales son multicapa, ya que la capacidad expresiva de una sola capa es muy limitada). Estas redes están compuestas por tres tipos de capas (Ilustración 17):

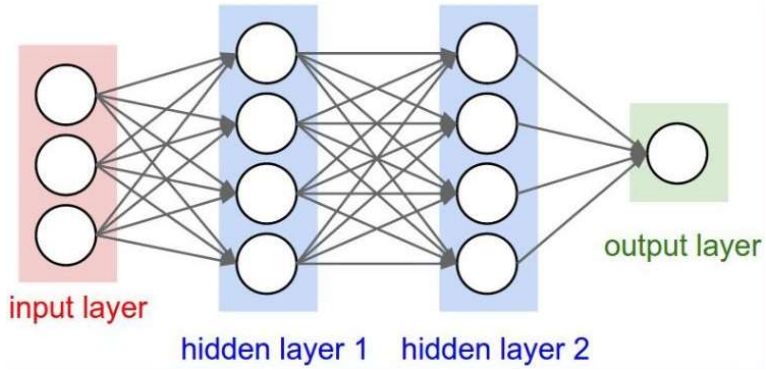


Ilustración 17: Tipos de capas de una red neuronal

- Una capa de entrada (**input layer**) que recibe los datos
- Una o más capas ocultas (**hidden layer**) en las que se realizan la mayor cantidad de los cálculos. A mayor número de capas ocultas, mayor complejidad del modelo, por lo que, si vemos que los resultados devueltos por este no son lo suficientemente buenos, podemos considerar como una opción viable aumentar el número de capas ocultas. También hay que tener en cuenta que, si incrementamos el número de capas ocultas en exceso, el modelo será más exigente computacionalmente sin ninguna necesidad, lo que supondrá un desperdicio de tiempo y recursos. En la práctica suele ser común que baste simplemente con una o dos capas ocultas.
- Una capa de salida (**output layer**) que devuelve los resultados finales calculados por el modelo. La estructura y el tipo de contenido de esta capa varía en función del tipo de predicción que se esté realizando. En tareas de regresión la red devolverá un valor numérico que puede representar desde la temperatura estimada en un día dados los datos del clima hasta el ángulo de giro de un vehículo automatizado dada una imagen con su posición actual. En tareas de clasificación, como la que nos ocupa en este proyecto, la capa de salida consistirá en un vector con un número de dimensiones igual al número de posibles categorías en las que puede clasificarse la entrada. El contenido de este vector será una distribución de probabilidades que representa la probabilidad de que la entrada pertenezca a cada una de las distintas clases. En este proyecto se está llevando a cabo una tarea de clasificación binaria (solo existen dos posibles categorías, noticia real o falsa), por lo que un único valor numérico como salida es suficiente. Este valor estará comprendido entre 0 y 1, si dicho valor es mayor o igual a 0.5, la entrada pertenecerá a una categoría, en caso contrario pertenecerá a la otra. En estos casos

también podría emplearse la distribución de probabilidades explicada anteriormente pero con un vector de dimensión 2.

Antes de profundizar más en el funcionamiento de una red completa, es necesario explicar el concepto de perceptrón.

4.8.2. Perceptrón

El perceptrón es el elemento más simple de la red neuronal, es cada uno de los nodos que componen a la red y se organizan en capas [42].

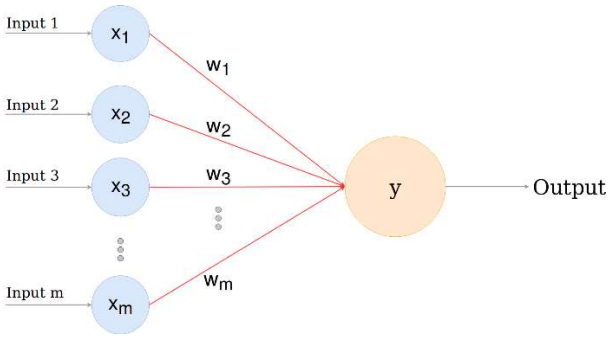


Ilustración 18: Perceptrón

El perceptrón recibe una serie de entradas consistentes en valores numéricos que pueden proceder de otras neuronas o de la capa de entrada y que se representan mediante aristas en el grafo de computación. Además de contener las entradas, estas aristas también tienen asignado un valor numérico conocido como peso. Dentro del perceptrón se realiza la combinación lineal de las entradas y los pesos. Una vez realizada esta combinación lineal, la utilizamos como parámetro para una función de activación, como puede ser la función sigmoide (Ecuación 4) [43].

$$f(x) = \frac{1}{1 + e^{-x}}$$

Ecuación 4: Función sigmoide

La función de activación es usualmente una abstracción que representa la potencial activación de una neurona. En su forma más simplificada, esta función es binaria, devolviendo simplemente 0 ó 1 en función de la entrada que reciba.

Utilizar estas funciones permite que las redes neuronales sean algo más que una simple combinación lineal de elementos, además de resolver problemas como el gradiente explosivo [44], al mantener los valores dentro de un rango.

Tras calcular la función de activación de la combinación lineal, obtenemos la salida del perceptrón, que puede ser la salida de la propia red neuronal o conectar con uno o varios perceptrones de la capa siguiente. En este último caso, la salida del perceptrón constituirá una de las entradas de uno o varios perceptrones, de nuevo cada una de estas entradas se representará como una arista en el grafo (cada una de ellas con su propio peso asignado, a pesar de tratarse de la misma salida).

La conexión de un gran número de perceptrones utilizados en capas es lo que dota a las redes neuronales de su gran potencia. La estrategia más común a la hora de conectar capas entre sí son las fully connected layers o capas densas, en la que cada uno de los perceptrones de cada capa se une mediante aristas con todos los perceptrones de la siguiente, esto se cumple con todas las capas hasta alcanzar la de salida.

El proceso mediante el cual, dada una entrada se calculan las funciones de activación de manera sucesiva capa por capa hasta llegar a la capa de salida se conoce como propagación hacia adelante. Una vez entrenada, una red neuronal se limita simplemente a realizar esta propagación para procesar nuevas entradas que no han sido utilizadas durante el entrenamiento.

Explicado el funcionamiento y estructura general de una red neuronal, nos falta por desarrollar como se realiza el entrenamiento del modelo para que sea capaz de realizar el proceso explicado previamente de manera correcta. Esto se lleva a cabo mediante el algoritmo de retropropagación.

4.8.3. Algoritmo de retropropagación

El algoritmo de retropropagación es lo que nos permite ajustar los valores de los pesos de las aristas del grafo de computación con el objetivo de maximizar el rendimiento del modelo, esta optimización de los pesos es en lo que consiste lo que se conoce popularmente como entrenamiento del modelo [45].

Para realizar este proceso, necesitamos un conjunto de entrenamiento anotado con el que entrenar el modelo. En el caso de este proyecto, dicho conjunto consistiría en una gran cantidad de noticias en formato textual y, para cada una de ellas, su categoría correspondiente (real o fake).

Al iniciar la red neuronal, los pesos se inicializan con unos valores aleatorios, por lo que las predicciones antes de entrenarla serán aleatorias. Lo que hace el algoritmo de retropropagación es emplear un ciclo de propagación-adaptación en dos fases [46]. Una vez que se aplica un patrón a la entrada de la red como estímulo (en nuestro caso, introducimos la representación de una noticia en la capa de entrada), este se propaga desde la primera capa hasta la salida de la red (esto es la propagación hacia adelante explicada en el apartado anterior). La salida obtenida se compara con la salida deseada (aquella con la que la entrada ha sido etiquetada en el conjunto de entrenamiento) y se calcula el error cometido empleando una función de pérdidas, como puede ser la entropía cruzada de sigmoide (Ecuación 5) [47].

$$CE(y, \hat{y}) = -\frac{1}{N}\sum_{i=0}^{N}(y * \log(\hat{y}_i) + (1 - y) * \log(1 - \hat{y}_i))$$

Ecuación 5: Entropía cruzada de sigmoide

En la ecuación anterior, y es la clase a la que pertenece la entrada (de acuerdo con el conjunto de datos), $\hat{y}$ es la clase predicha por el modelo y N representa el número de clases.

Las salidas de error se propagan hacia atrás capa por capa, partiendo desde la capa de salida, cada neurona recibe una fracción del total del error cometido en base a la contribución relativa de su respectiva salida a la salida final. De esta forma, los pesos se ajustan para maximizar el rendimiento del modelo en el conjunto de entrenamiento (Entendemos que el rendimiento del será mayor cuanto menor sea el valor devuelto por la función de pérdidas).

A la hora de llevar a cabo el entrenamiento del modelo, suele ser común recorrer todo el conjunto de entrenamiento más de una vez para que los pesos se ajusten correctamente. A cada una de estas iteraciones sobre el conjunto de entrenamiento se les conoce como épocas. Elegir un número adecuado de épocas es imprescindible para evitar problemas de Underfitting y Overfitting (estos dos conceptos se explican con mayor detalle al inicio de la sección métricas utilizadas).

Es importante hacer énfasis en la distinción entre dos conceptos que en muchas ocasiones suelen utilizarse indistintamente y de forma errónea en la práctica, lo que puede generar confusión. Estos dos conceptos son los parámetros y los hiperparámetros del modelo [48].

Los parámetros de un modelo son aquellas variables internas de este y cuyos valores pueden estimarse a partir de los datos durante el entrenamiento. El ejemplo más característico de parámetro son los pesos de las redes neuronales, cuyos valores son optimizados gracias al algoritmo de retropropagación.

Los hiperparámetros del modelo son variables cuyo valor no puede ser estimado a partir de los datos, estos valores son escogidos antes de iniciar el entrenamiento. Algunos hiperparámetros en los modelos neuronales son: el número de capas, la cantidad de neuronas por capa, la función de activación, la función de pérdidas, el ratio de dropout, entre todos. La optimización de hiperparámetros es un paso imprescindible para sacarle el máximo provecho a un modelo neuronal. Este paso se realizará a lo largo del sprint 4.

4.8.4. Ratio de dropout

A la hora de entrenar modelos neuronales, suele ser común aplicar dropout [49] a las capas densas. Esta técnica consiste en ignorar ciertas neuronas de la capa en cuestión durante el entrenamiento de una entrada. Esta capa toma como hiperparámetro un ratio de dropout cuyo valor recomendado oscila entre 0 y 0.5, este ratio es la probabilidad de ignorar cada una de las neuronas de la capa densa.

Utilizar dropout nos permite reducir la interdependencia entre las neuronas de una capa durante el entrenamiento, lo que provoca que la red neuronal se vea obligada a aprender patrones más robustos que son útiles en conjunción con varios subconjuntos de neuronas. Otra ventaja de aplicar esta técnica es que ayuda a prevenir el Overfitting. No obstante, también provoca que la convergencia sea algo más lenta.

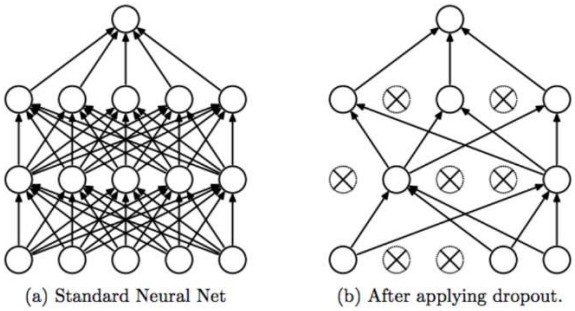


Ilustración 19: Comparación entre red neuronal estándar y red con dropout de 0.5

4.8.5. Métricas utilizadas

En este apartado se explican las métricas que se emplearán para evaluar los distintos modelos a aplicar. Dichas métricas nos permitirán realizar una comparación cuantitativa entre modelos. Todas estas métricas se computarán tanto en el conjunto de entrenamiento como en el de pruebas.

Calculando las métricas en el conjunto de entrenamiento podremos comprobar si el modelo en cuestión se adapta correctamente a este conjunto. Puede ocurrir que no logre ajustarse, devolviendo malos resultados incluso en el entrenamiento, este fenómeno se conoce como Underfitting y suele significar que se requieren más datos o cambiar la configuración del modelo. Algunas formas de modificar el modelo son:

- Realizar un ajuste de hiperparámetros
- Cambiar la distribución de las capas (en el caso de las redes neuronales)
- Elegir otra función de optimización
- Elegir otra función de pérdidas
- Mejorar el preprocesado de los datos

En el caso de que el modelo se ajuste demasiado bien al conjunto de entrenamiento, podemos estar ante un caso de Overfitting, o sobreajuste. El Overfitting se da cuando un modelo da muy buenos resultados en el conjunto de entrenamiento, pero resultados muy mediocres en el de pruebas. Esto se debe a que, en lugar de entrenar un modelo capaz de generalizar y encontrar patrones en los datos utilizados para entrenar, este empieza a tener en cuenta ruido y fluctuaciones aleatorias para ajustarse el conjunto de entrenamiento, lo que tiene repercusiones negativas a la hora de clasificar nuevos datos. [50]

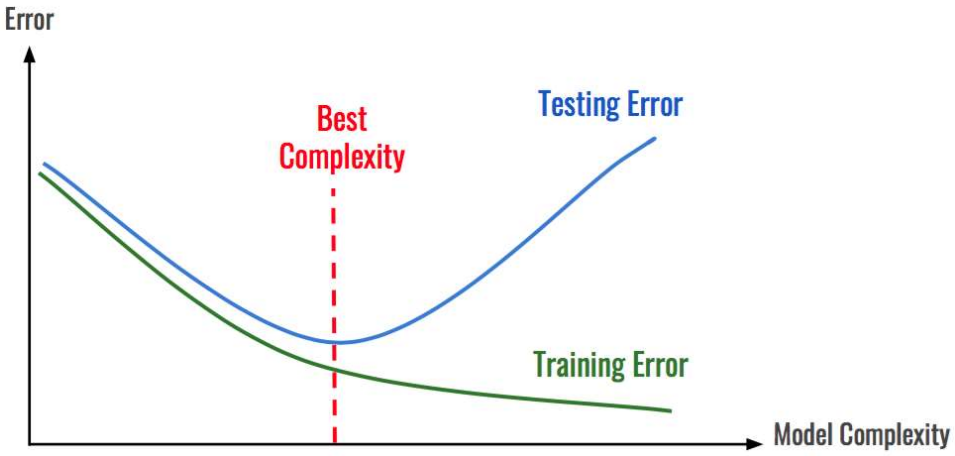


Ilustración 20: Ejemplo de Overfitting

Representaciones gráficas de la evolución de los resultados en ambos conjuntos como la de la Ilustración 20 nos permiten sacar conclusiones sobre cómo se comporta el modelo a lo largo de las distintas épocas tanto en el conjunto de entrenamiento como en el de pruebas, además de determinar si son necesarias más o menos épocas o realizar un ajuste de hiperparámetros, obteniendo de esta forma todo el potencial que ofrecen los distintos métodos de clasificación, además de elegir los parámetros a cambiar durante la fase de parameter tuning de manera más intuitiva.

Generalmente los modelos son evaluados observando el número de instancias clasificadas correcta e incorrectamente. En el caso de una clasificación binaria, como la que se está llevando a cabo en el presente proyecto, pueden darse los siguiente cuatro casos:

- **True Positive (TP):** La clase positiva se predice correctamente (En este proyecto, la clase positiva es una fake new). Es el caso que se da cuando una fake new es clasificada como tal por el modelo.
- **True Negative (TN):** La clase negativa se predice correctamente. Esto ocurre cuando una noticia fiable es clasificada como tal por el modelo.
- **False Positive (FP):** La clase positiva se predice incorrectamente (Una fake new es clasificada como fiable).
- **False Negative (FN):** La clase negativa se predice incorrectamente (Una noticia fiable es clasificada como fake new).

La distribución de estos valores suele expresarse mediante una matriz de confusión como la de la Tabla 11:

		Clase Predicha	
		Fake New (Positivo)	**Noticia Fiable (Negativo)**
Clase Real	**Fake New (Positivo)**	TP	FN
	Noticia Fiable (Negativo)	FP	TN

Tabla 11: Matriz de Confusión

A continuación, se describen las métricas utilizadas para evaluar los modelos:

4.8.5.1. Accuracy (Exactitud)

La Accuracy mide el porcentaje de casos que el modelo ha logrado predecir correctamente. Tal y como se muestra en la Ecuación 6, su valor se estima como el cociente de la suma de los True Positives y los True Negatives (Predicciones correctas) con el total de predicciones realizadas. [51]

$$acc = \frac{TP + TN}{TP + TN + FP + FN}$$

Ecuación 6: Accuracy

El principal problema de esta métrica es que puede resultar engañosa en función de la distribución de clases que exista en el conjunto de datos. Por ejemplo, si estuviéramos intentando entrenar un modelo para detectar la presencia de un virus poco común en distintos pacientes en base a sus síntomas, es de esperar que la mayoría de las instancias del dataset sean negativas (el paciente está sano). Si suponemos que el dataset está compuesto por datos de 1100 pacientes de los cuales 1000 están sanos y 100 están enfermos, el modelo entrenado podría predecir a casi todos los pacientes como sanos y aun así obtener una buena Accuracy. Si suponemos que el modelo ha realizado una clasificación como la de la matriz de confusión de la Tabla 12:

		Clase Predicha	
		Enfermo (Positivo)	**Sano (Negativo)**
Clase Real	**Enfermo (Positivo)**	1	99
	Sano (Negativo)	0	1000

Tabla 12: Ejemplo de matriz de confusión

Si calculamos el valor de la Accuracy de la matriz de confusión previa, obtendríamos un valor del 91% (Ecuación 7), por lo que, en base a este resultado, podríamos concluir que estamos ante un buen modelo, a pesar de que está clasificando a casi todos los pacientes como sanos.

$$acc = \frac{1 + 1000}{1 + 99 + 1000 + 0} = 0.91$$

Ecuación 7: Ejemplo de cálculo de Accuracy

El caso mostrado en el párrafo anterior es un ejemplo extremo, y una situación como esa no se daría en este proyecto, ya que las clases han sido balanceadas. No obstante, esta métrica no nos permite analizar si el modelo se decanta más por una clase que por otra, por lo que en ocasiones no nos provee información suficiente.

4.8.5.2. Precision (Precisión)

La precisión es el número de ejemplos de la clase positiva clasificados correctamente (True Positives) divididos por el número total de ejemplos que han sido clasificados como positivos (True Positives y False Positives). [51]

$$p = \frac{TP}{TP + FP}$$

Ecuación 8: Precision

Por ejemplo, en el ejemplo expuesto anteriormente, el valor de la precisión sería el siguiente, de acuerdo con la matriz de confusión de la Tabla 12:

$$p = \frac{TP}{TP + FP} = \frac{1}{1 + 0} = 1$$

Ecuación 9: Ejemplo de cálculo de Precision

De esta forma obtenemos una precisión del 100%, ya que solo clasificamos una muestra positiva correctamente y ningún ejemplo negativo se ha clasificado correctamente.

4.8.5.3. Recall (Exhaustividad)

El Recall es el número de ejemplos de la clase positiva clasificados correctamente (True Positives) divididos por el número total de ejemplos positivos (True Positives y False Negatives) [51].

$$r = \frac{TP}{TP + FN}$$

Ecuación 10: Recall

En el ejemplo de la Tabla 12, esta métrica se calcularía de la siguiente forma:

$$r = \frac{TP}{TP + FN} = \frac{1}{1 + 99} = 0.01$$

Ecuación 11: Ejemplo de cálculo de Recall

En un caso como el del ejemplo, el Recall sería de un 1%. Si interpretamos los resultados devueltos por esta métrica y los contrastamos con los devueltos por la precisión, nos damos cuenta de que el modelo ofrece muy buenos resultados prediciendo la clase negativa (sano) pero resultados pésimos prediciendo la clase positiva (enfermo). Ambas métricas por separado no nos aportan dicha información, por lo que es necesario calcular ambas para poder realizar un correcto análisis del comportamiento del modelo. No obstante, necesitamos una medida que combine precisión y recall.

4.8.5.4. F1-Score

Aunque la precisión y el recall de un modelo nos permiten analizar su comportamiento, resulta muy poco intuitivo comparar dos clasificadores distintos utilizando dos métricas. Por esta razón, la medida utilizada para realizar dicha comparación será F1-Score, que se calcula como la media armónica entre precisión y recall, como se muestra en la Ecuación 12: [51]

$$F_1 = \frac{2 \cdot p \cdot r}{p + r}$$

Ecuación 12: F1-Score

De esta forma, para que F1 nos devuelva un buen valor, tanto los valores de la precisión como del recall deben ser buenos. En el ejemplo de la Tabla 12:

$$F_1 = \frac{2 \cdot p \cdot r}{p + r} = \frac{2 \cdot 1 \cdot 0.01}{1 + 0.01} = 0.0198$$

Ecuación 13: Ejemplo de cálculo de F1-Score

El valor de F1 sería cercano al 2%, lo que resume en una sola métrica que se trata de un mal modelo.

PARTE III. SISTEMA DESARROLLADO

5. Diseño inicial

5.1. Objetivos

El principal objetivo de este proyecto es aplicar diversos algoritmos de Procesamiento del Lenguaje Natural combinados con técnicas de Deep Learning para la detección de fake news en un corpus, realizando una comparación entre los resultados ofrecidos por cada uno de ellos para así determinar cuál es el más apropiado para resolver el problema, indicando siempre las distintas ventajas e inconvenientes de cada método.

Para la aplicación de estos algoritmos y para la creación de la Red Neuronal Recurrente, se usarán diversas librerías de código abierto en un entorno que correrá sobre Python, algunas de estas librerías son sklearn, nltk, y tensorFlow, entre otras. Para el preprocesado y tratamiento de datos se emplearán librerías como pandas. Todas estas tecnologías son descritas con mayor detalle en la sección Tecnologías.

Por tanto, el objetivo general del proyecto es el siguiente:

OBJETIVO
Desarrollar un sistema de detección de fake news basado en el uso de Redes Neuronales Artificiales, realizando un estudio comparativo entre las distintas técnicas empleadas.

Tabla 13: Objetivo general del Proyecto

5.2. Requisitos

Cada uno de estos requisitos (a excepción de los que aparecen en Tabla 14 y Tabla 15) se refieren a la aplicación de un método para clasificar el conjunto de datos. Para cada uno de estos métodos se seguirán los siete pasos para el Machine Learning propuestos por Google [4] de elección, entrenamiento y evaluación del modelo, junto con el ajuste de parámetros y la predicción.

Asimismo, los requisitos REQ-01 (Tabla 14) y REQ-02 (Tabla 15) abarcan los pasos de adquisición del conjunto de datos y preparación de los datasets respectivamente.

Todos y cada uno de estos requisitos deben ser documentados adecuadamente, justificando siempre las decisiones tomadas y mostrando los resultados obtenidos.

REQ-01: Adquisición de datos
Seleccionar un conjunto de datos amplio y de calidad

Tabla 14: REQ-01

REQ-02: Preprocesado del Conjunto de Datos
Realizar un preprocesado del conjunto de datos que permita mejorar el rendimiento de los métodos que se van a aplicar.

Tabla 15: REQ-02

REQ-03: Bag of Words – Binary Counts
Aplicar Binary Words usando Binary Counts para la clasificación de los datos, siguiendo los principios de elección, entrenamiento y evaluación del modelo.

Tabla 16: REQ-03

REQ-04: Bag of Words – Term Frequency
Aplicar Binary Words usando Term Frequency para la clasificación de los datos, siguiendo los principios de elección, entrenamiento y evaluación del modelo.

Tabla 17: REQ-04

REQ-05: Bag of Words – Tf-idf
Aplicar Binary Words usando Term Frequency e Inverse Document Frequency para la clasificación de los datos, siguiendo los principios de elección, entrenamiento y evaluación del modelo.

Tabla 18: REQ-05

REQ-06: Word2Vec – Skipgram
Aplicar Word2Vec usando Skipgram para la clasificación de los datos, siguiendo los principios de elección, entrenamiento y evaluación del modelo.

Tabla 19: REQ-06

REQ-07: Word2Vec – CBOW
Aplicar Word2Vec usando Continuous Bag Of Words (CBOW) para la clasificación de los datos, siguiendo los principios de elección, entrenamiento y evaluación del modelo.

Tabla 20: REQ-07

REQ-08: Doc2Vec – DM
Aplicar Doc2Vec usando Distributed Memory (DM) para la clasificación de los datos, siguiendo los principios de elección, entrenamiento y evaluación del modelo.

Tabla 21: REQ-08

REQ-09: Doc2Vec – DBOW
Aplicar Doc2Vec usando Distributed Bag Of Words (DBOW) para la clasificación de los datos, siguiendo los principios de elección, entrenamiento y evaluación del modelo.

Tabla 22: REQ-09

REQ-10: Red Neuronal Recurrente
Implementar una Red Neuronal Recurrente para la clasificación de los datos, siguiendo los principios de elección, entrenamiento, evaluación del modelo y ajuste de parámetros.

Tabla 23: REQ-10

REQ-11: Optimización de Hiperparámetros
Optimizar los hiperparámetros de la Red Neuronal Recurrente

Tabla 24: REQ-11

5.3. Arquitectura

5.3.1. Arquitectura lógica

La Ilustración 21 muestra un diagrama de componentes que describe el comportamiento de cada uno de los distintos componentes que forman el sistema. Este diagrama se cumple para cada uno de los modelos a implementar.

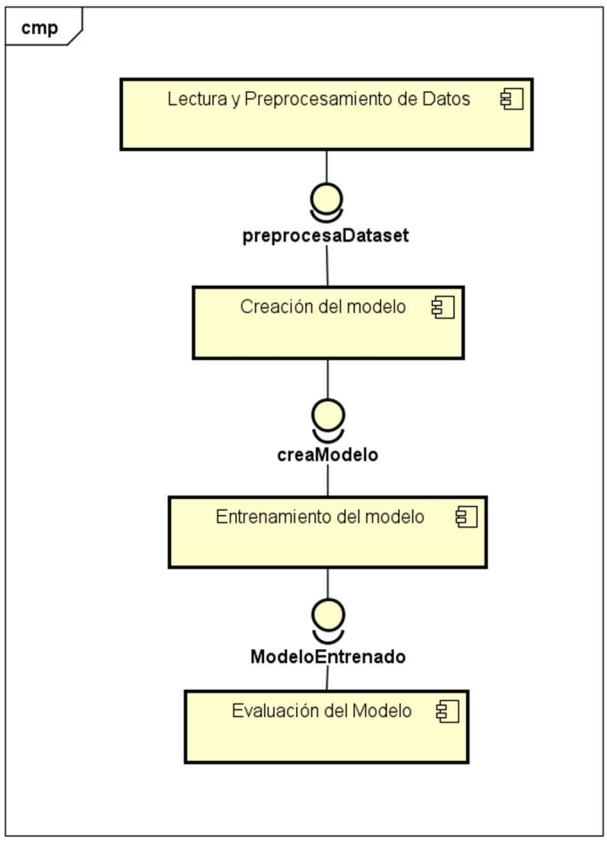


Ilustración 21: Diagrama de Componentes

- **Lectura y preprocesamiento de datos:** Este componente se encarga de leer y procesar el conjunto de datos, para que este tenga la mayor calidad posible de cara al entrenamiento del modelo. También divide el conjunto de datos procesado en subconjunto de entrenamiento y subconjuntos de pruebas.
- **Creación del modelo:** Crea un modelo que pueda entrenarse empleando el conjunto de datos devuelto por el componente anterior.
- **Entrenamiento del modelo:** El modelo creado por el componente previo es entrenado utilizando el subconjunto de entrenamiento.

- **Evaluación del modelo:** Se realiza una evaluación del modelo empleando el subconjunto de pruebas para comprobar el rendimiento. Las métricas empleadas para evaluar los distintos modelos serán Accuracy, Precision, Recall y F1-Score. Estas métricas han sido calculadas también en el conjunto de entrenamiento. El análisis de estas mediciones en ambos conjuntos nos permite determinar si hay problemas de overfitting o de underfitting.

5.3.2. Diagrama de clases

El código que se desarrollará durante este proyecto seguirá el paradigma de la programación funcional [52] y no el de la programación orientada a objetos. No obstante, se emplearán librerías que trabajan con clases, representadas en el diagrama de la Ilustración 22, este diagrama es una abstracción inicial de la estructura del proyecto que sufrirá diversas modificaciones y ampliaciones a lo largo de los sprints. Este diagrama se cumple para cada uno de los modelos a implementar.

Las clases de color amarillo son las que van a ser implementadas a lo largo del proyecto, mientas que las azules representan las librerías empleadas y sus respectivas funciones, esta nomenclatura se sigue en todos los diagramas de clases del documento.

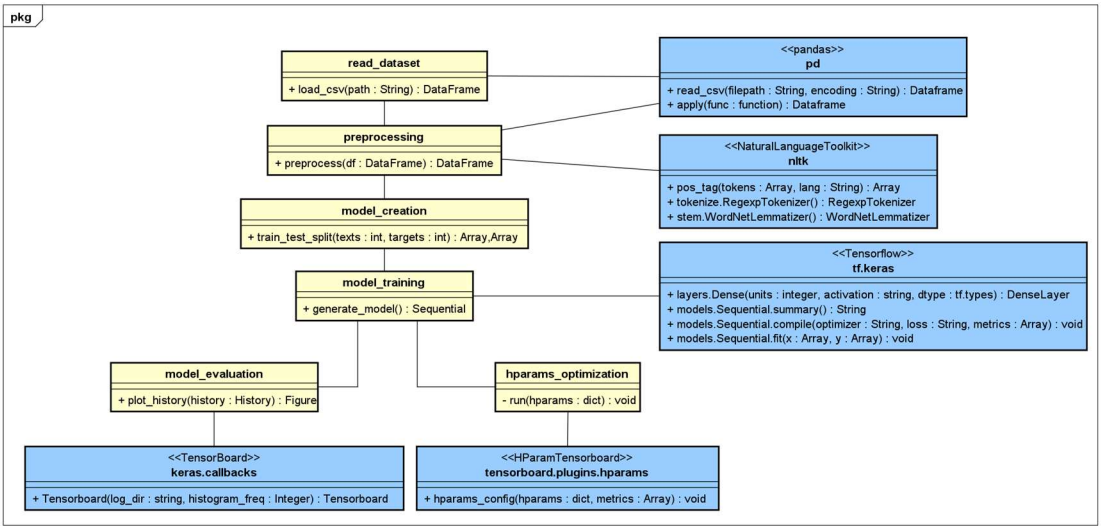


Ilustración 22: Diagrama de clases

Las funciones básicas indicadas en la Ilustración 22 son:

- **Read_dataset:** Realiza la lectura del archivo csv empleando la librería pandas, obteniendo un objeto de tipo DataFrame, que contiene el dataset sin preprocesar.
- **preprocessing:** Encargada del preprocesamiento del dataset para facilitar el entrenamiento del modelo y mejorar los resultados de este. Una vez preprocesado, el dataset se divide en conjunto de entrenamiento y pruebas.
- **Model_creation:** Preparará los datos para que puedan usarse como entrada para el modelo en cuestión

- **Model_training:** Creará un modelo en TensorFlow con una estructura determinada y lo entrenará.
- **Model_evaluation:** Se utiliza el modelo entrenado en el paso anterior para clasificar el conjunto de pruebas. Una vez realizada la clasificación, se calculan las métricas correspondientes (accuracy, precision, recall y F1-score).
- **Hparams_optimization:** Esta clase contiene las funciones que permiten optimizar la Red Neuronal Recurrente utilizando TensorBoard, permitiendo sacarle el máximo partido al modelo.

5.3.3. Diagrama de secuencia

El diagrama de Secuencia muestra la ejecución temporal del diagrama de clases realizado en el apartado previo. Este diagrama se muestra en la Ilustración 23 y se cumple para todos los modelos de predicción que se van a implementar, representando el proceso completo de creación, entrenamiento y evaluación del modelo, además del preprocesamiento del conjunto de datos.

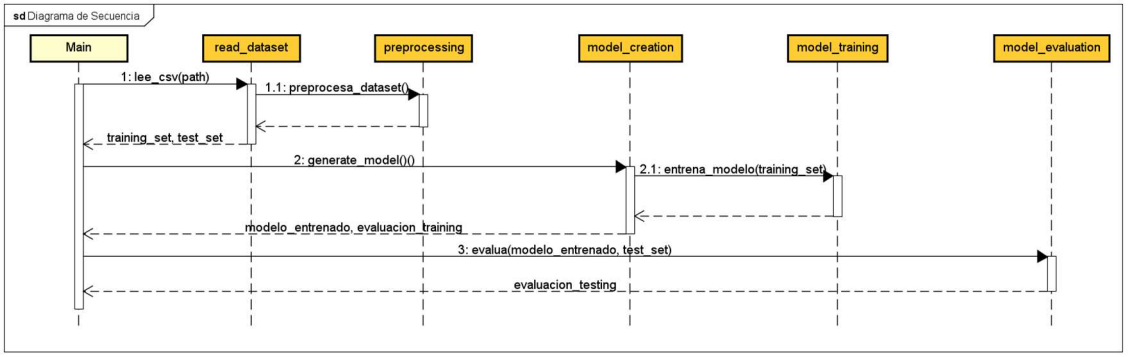


Ilustración 23: Diagrama de Secuencia

El orden que se sigue es el siguiente:

1. Lectura del fichero csv
   1.1. Preprocesado del dataset, devuelve el dataset preprocesado y dividido en conjunto de entrenamiento y pruebas
2. Creación del modelo
   2.1. Entrenamiento del modelo, toma como parámetro el conjunto de entrenamiento y devuelve el modelo entrenado y la evaluación de los resultados en el conjunto de entrenamiento.
3. Evaluación del modelo entrenado usando el conjunto de pruebas y el modelo entrenado. Devuelve la evaluación del modelo.

5.4. Sprints

Inicialmente, se consideró la idea de hacer un sprint por cada uno de los puntos de *Los 7 pasos del machine Learning.* Sin embargo, dado que se van a aplicar diversos métodos, cada uno de los cuales sigue estos mismos pasos, se ha tomado la decisión de dedicar el primer sprint a la adquisición y preparación del conjunto de datos (además de generar la documentación básica

del proyecto, junto con la planificación inicial), ya que seleccionar un conjunto de datos apropiado y procesarlo adecuadamente es un punto de partida fundamental para que los métodos que se aplicarán posteriormente den buenos resultados.

En lo que respecta al resto de sprints, en el segundo se aplicarán una serie de baselines para poder realizar un estudio comparativo con algoritmos más avanzados, además de otros métodos de mayor complejidad (Word2Vec y Doc2Vec).

El tercer sprint se dedicará por completo al desarrollo de una red neuronal recurrente, que es el método de mayor complejidad de los seleccionados, y el que requerirá más tiempo para ser implementado y entrenado. Para cada uno de los modelos, se realizarán los pasos de elección, entrenamiento y evaluación del modelo. La decisión de organizar los sprints de esta forma, en lugar de dedicar un sprint a realizar solamente uno o más de los pasos del Machine Learning se ha tomado por las razones que se exponen a continuación:

- Siguiendo ese orden, no tendríamos resultados de ninguno de los métodos hasta el último sprint, por lo que no podría realizarse ninguna comparación entre métodos hasta el final.
- Alternar entre varios métodos distintos de forma constante, teniendo en cuenta lo diferentes que son entre sí puede generar muchas confusiones y errores, especialmente si no se tiene experiencia previa.
- Los métodos por emplear se han ordenado para ejecutarlos de menor a mayor complejidad, esto permite que se realice un proceso de aprendizaje incremental en el que la dificultad va aumentando por cada nuevo método, pudiendo aplicar los conocimientos adquiridos previamente para resolver problemas cuya dificultad va incrementando.

En el cuarto sprint se llevará a cabo el análisis final de los resultados obtenidos, realizando comparaciones entre los distintos métodos, y se redactarán las conclusiones del proyecto. El desglose de horas para cada sprint se muestra en la Tabla 25.

Sprint	Descripción	Horas
1	Adquisición y Preprocesado de datasets	40
2	Aplicación de Baselines	60
3	Implementación de Red Neuronal Convolucional	60
4	Ajuste de parámetros y análisis de los resultados	60

Tabla 25: Sprints del proyecto

6. Sprint 1: Adquisición y preprocesado de datasets

Este primer sprint tiene como objetivos encontrar, analizar y procesar el conjunto de datos que se utilizará para entrenar y probar los distintos modelos predictivos a implementar en los futuros sprints. Este es el primero de los pasos del aprendizaje automático y tiene una importancia vital, ya que la correcta elección y preparación del conjunto de datos determinará en gran medida la calidad del modelo desarrollado. Este sprint tiene una duración de 19 días, con un total de 40 horas.

6.1. Requisitos

Los requisitos que se abordarán en este sprint son los siguientes:

REQ-01: Adquisición de datos
Seleccionar un conjunto de datos amplio y de calidad

Tabla 26: REQ-01

REQ-02: Preprocesado del Conjunto de Datos
Realizar un preprocesado del conjunto de datos que permita mejorar el rendimiento de los métodos que se van a aplicar.

Tabla 27: REQ-02

6.2. Diseño

6.2.1. Diagrama de clases

El diagrama de clases a implementar en este sprint es el mostrado en la Ilustración 24. El diagrama se centra en las clases read_dataset y preprocessing.

La clase read_dataset se encarga de llevar a cabo la lectura del fichero .csv que contiene el conjunto de datos, convirtiéndolo en un objeto de tipo dataframe, todo ello empleando la librería pandas.

La clase preprocessing recibe el conjunto de datos en formato dataframe y le aplica la función de preprocesado, que prepara los datos para su posterior clasificación.

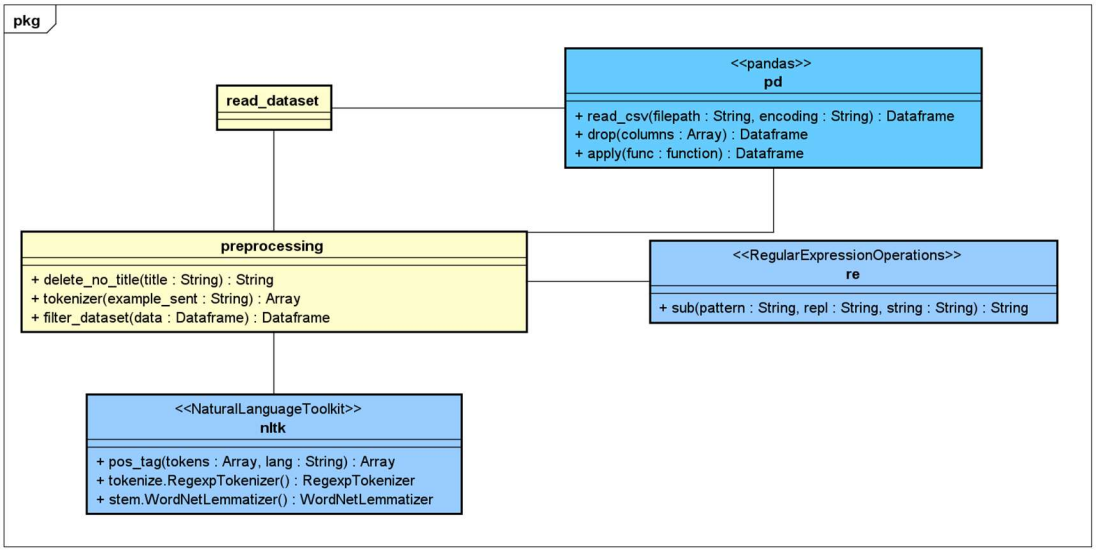


Ilustración 24: Diagrama de clases Sprint 1

6.2.2. Análisis del conjunto de datos

El dataset empleado para realizar la clasificación es TI-CNN dataset [53]. Se trata de un conjunto de datos de código abierto compuesto por un total de 20.015 noticias, de las cuales 11.941 están clasificadas como falsas y las 8.074 restantes como reales.

Este conjunto de datos ha sido creado por un grupo de investigadores cuyos miembros pertenecen a universidades de Pekín (Beihang University), Chicago (University of Illinois) y Florida (Florida State University) [1]. Las noticias falsas que lo componen han sido obtenidas del dataset de Kaggle "Getting Real About Fake News" [54] que contiene noticias publicadas a finales del año 2016. Las noticias reales han sido extraídas de fuentes conocidas, como el New York Times y el Washington Post.

Para cada artículo, el dataset contiene información adicional como el autor, el país, el idioma de la noticia y la url de la imagen, entre otros datos. El enfoque de este proyecto está basado en técnicas de procesamiento del lenguaje natural, por lo que solo se utilizará la información textual. Un artículo publicado unos meses después que el artículo original [55] consiguió obtener mejores resultados que la publicación original empleando solamente el texto, por lo que podemos considerar el texto como información suficiente para satisfacer los objetivos de este proyecto.

El artículo original ha sido citado por otras 33 publicaciones [56], por lo que se puede asumir que la calidad de los datos es adecuada para entrenar modelos neuronales avanzados. También se emplearán modelos más básicos para analizar el comportamiento de los datos y detectar posibles sesgos, además de realizar un estudio comparativo entre los distintos métodos de clasificación empleados en base a su rendimiento.

Para la realización de este proyecto, solo se utilizarán los campos text, title y Type, donde text es el contenido de la noticia, title es el título de la esta y Type es la clasificación que recibe. La variable type toma los valores "real" y "fake".

La siguiente gráfica muestra la distribución de los tipos de Noticias en el corpus:

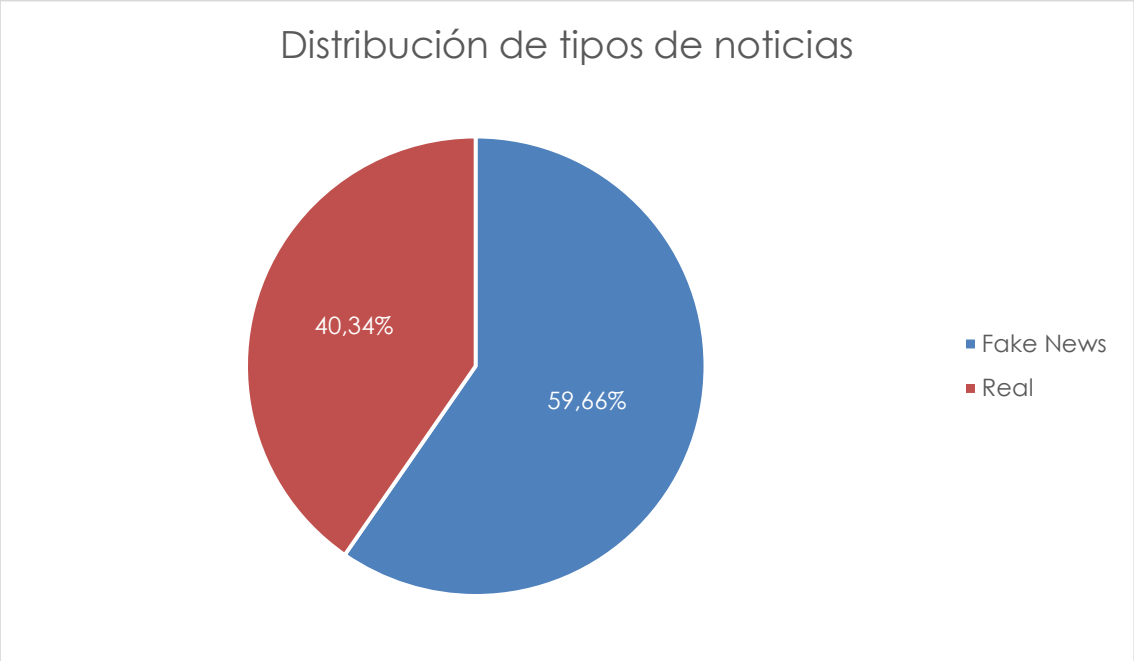


Ilustración 25: Distribución de tipos de noticias en TI-CNN dataset

El 59,66% de las noticias del Corpus son fake, mientras que el 40,34% restante son noticias reales.

6.2.3. Preprocesamiento

La principal labor de implementación de este sprint es el preprocesamiento del conjunto de datos. Este paso es imprescindible en procesamiento del lenguaje natural para eliminar ruido y mejorar considerablemente el rendimiento de los algoritmos de clasificación que se aplicarán posteriormente.

A la hora de llevar a cabo el preprocesamiento de un conjunto de datos, es importante tener en cuenta que convertir los datos a un formato que sea entendible por un algoritmo de Machine Learning los lleva a un estado generalizado que puede implicar cierta pérdida de fidelidad con respecto a los datos originales, lo que en ocasiones puede suponer una pérdida de información. Por esta razón, resulta vital conocer las ventajas e inconvenientes de cada técnica antes de decidir si aplicarla o no.

Las técnicas de preprocesamiento dentro del procesamiento del lenguaje natural suelen clasificarse dentro de las siguientes cuatro categorías [57]:

- **Limpieza:** Consiste en eliminar las partes menos útiles del texto mediante la aplicación de técnicas como Stopwords removal (consistente en eliminar palabras que aportan poca o ninguna información, como preposiciones y phrasal verbs), tratar con el uso de mayúsculas y caracteres especiales, entre otros.
- **Anotación:** Técnicas relacionadas con extraer información sobre la estructura de los textos. El ejemplo más claro es Part-Of-Speech Tagging (esta técnica asocia a cada palabra su respectiva categoría gramatical en el contexto en el que ha sido empleada).

- **Normalización:** Esta categoría abarca varias técnicas de traducción, mapeado y estandarización, como Word Stemming (reduce cada palabra a su raíz o lexema) y Lemmatization (elimina las inflexiones de las palabras).
- **Análisis:** Consiste en manipular y generalizar estadísticamente el conjunto de datos para inferir características de este o realizar un análisis de dichas características.

Los métodos de preprocesamiento seleccionados han sido:

1. **Capitalization:** Consiste en convertir todo el texto a minúsculas. Es bastante común encontrarnos con letras mayúsculas en un texto, para indicar nombres propios o el comienzo de una oración. Muchas de las funciones que van a aplicarse posteriormente son case sensitive, lo que puede provocar que dos apariciones de una misma palabra se consideren como dos palabras completamente distintas simplemente porque en una de dichas apariciones la palabra empiece por mayúscula, esto puede afectar considerablemente al rendimiento de los algoritmos de aprendizaje automático.
   El principal inconveniente de esta técnica es que ciertas palabras tienen un significado distinto en función de si aparecen en mayúscula o en minúscula. Un ejemplo de esto es la palabra "us" en inglés, que en mayúscula se refiere a los Estados Unidos de América, mientras que en minúscula es un pronombre que indica primera persona del plural.
2. **Eliminar etiquetas html, direcciones email y urls:** Eliminación de ruido mediante el empleo de expresiones regulares. Es una de las técnicas de preprocesamiento más comunes, suele aplicarse siempre. Las etiquetas html se eliminan porque pueden aparecer en conjuntos de datos obtenidos mediante Web Scraping, como en el caso que nos ocupa.
3. **Eliminar signos de puntuación:** Este paso sigue el mismo principio que el anterior, se emplean expresiones regulares para eliminar los signos de puntuación. En caso de que no se aplicase esta técnica, cada signo de puntuación se consideraría como una palabra.
4. **POS Tagging:** Esta técnica devuelve la categoría sintáctica de cada una de las palabras del texto. Para determinar dicha categoría, no solamente se tiene en cuenta la categoría más común de la palabra en cuestión, sino también su contexto, por lo que los resultados obtenidos suelen ser muy precisos.
   La razón por la que se aplica esta técnica es porque la categoría sintáctica de las palabras es un parámetro para la función que se aplica en el próximo paso.
5. **Lemmatization:** [58] Esta técnica elimina las inflexiones de las palabras, convirtiendo los verbos en infinitivo y los sustantivos en singular. En lugar de aplicar heurísticas para recortar las palabras hasta quedarse con su supuesta raíz, esta técnica utiliza la base de datos léxica de WordNet [59] para eliminar las inflexiones de manera mucho más efectiva que otras técnicas alternativas. El único inconveniente de esta técnica es que requiere la categoría sintáctica de cada palabra como parámetro (en caso contrario considera a la palabra como un sustantivo), por lo que ha sido necesario aplicar POS-Tagging. Aunque este paso previo puede ralentizar ligeramente la velocidad de preprocesamiento, los buenos resultados ofrecidos por Lemmatization compensan enormemente dicha ralentización.
6. **Stop Words Removal:** Muchas de las palabras que forman parte de una oración o de un texto no tienen significado propio, sino que actúan como conectores entre palabras que

sí lo tienen. Algunas de estas palabras son las preprosiciones o los phrasal verbs. Stop Words Removal elimina dichas palabras del texto, lo que hace que el rendimiento de las distintas técnicas de Machine Learning mejore considerablemente.

Además de estas técnicas, también se consideró aplicar normalización de sinónimos, que unifica varias palabras como una sola si estas tienen una relación de sinonimia. Aunque esta técnica puede resultar útil para proyectos relacionados con extracción de información o relaciones entre documentos, se ha llegado a la conclusión de que en el dominio de este problema esta técnica puede conllevar una excesiva simplificación ya que, en el mundo del periodismo hay palabras que, si bien pueden considerarse sinónimos, pueden tener ciertas connotaciones o pertenecer a diferentes niveles del lenguaje que nos permitan inferir el rigor periodístico de la noticia.

Como alternativa a Lemmatization, se pensó en utilizar Word Stemming, una técnica más simple que emplea heurísticas basadas en reglas para eliminar cierto número de caracteres al final de una palabra con el objetivo de obtener la raíz de esta. El problema es que dichas heurísticas no funcionan demasiado bien y suelen cometer muchos errores, especialmente en el caso de los verbos irregulares y los sustantivos con una forma plural irregular.

6.3. Implementación

6.3.1. Análisis del conjunto de datos

Para realizar el análisis del dataset se ha utilizado la librería pandas de Python, diseñada para el análisis y la manipulación de datos.

El conjunto de datos contiene un total de 54 columnas con información adicional sobre cada noticia, como la fecha de publicación, la url de una imagen asociada, el país de la noticia, el autor o el idioma, entre otros. Para la realización de este proyecto solo nos interesan las columnas "title", "text" y "type".

No obstante, conviene realizar un análisis de algunas de estas propiedades para familiarizarnos con los datos. En este caso, las columnas que resulta más interesante analizar son "country" y "language", para saber si todas las noticias están en inglés, además de "site_url", para hallar las fuentes de las noticias reales.

6.3.1.1. Análisis de las columnas "Country" y "Language"

Tener noticias pertenecientes a distintos países puede resultar beneficioso, ya que permitiría a los modelos neuronales entrenarse en una mayor cantidad de ámbitos informativos, extrayendo patrones característicos de las fake news independientemente del tema de la noticia.

Sin embargo, la función de preprocesado y muchos de los métodos a aplicar están diseñados para textos en inglés (o en un solo idioma, ya que aumentar el número de idiomas sería inviable con la cantidad de datos de la que se dispone), lo que provocaría que el modelo final se viera obligado a clasificar aleatoriamente los textos escritos en idiomas distintos al inglés. También podría ocurrir, en el peor de los casos, si se da la situación de que los textos en otro idioma se encuentran en una sola categoría, que el algoritmo se entrenase de tal forma que clasificase las noticias escritas en otros idiomas en una categoría determinada, independientemente de su contenido.

Por estos motivos, es importante analizar qué cantidad de noticias han sido redactadas en un idioma distinto al inglés.

En el dataset completo, las noticias están en 17 idiomas distintos, entre los que se encuentra el inglés, esto lo sabemos gracias a la función unique de pandas (Ilustración 26).

```
print('Número de países: ' + str(len(df.country.unique())) + ' (también se incluye la etiqueta nocountry)\n')
print(df.country.unique())

# Podemos observar que las noticias están en 17 idiomas distintos, nos quedaremos solo con las noticias en inglés
print('\nNúmero de idiomas: ' + str(len(df.language.unique())) + ' Entre los que se incluye nan')
df.language.unique()
```

```
Número de países: 24 (también se incluye la etiqueta nocountry)

['US' 'CO' 'DE' 'GB' 'CA' 'AU' 'FR' 'EU' 'NL' 'LI' 'SG' 'ME' 'TV' 'ES'
 'RU' 'IN' 'nocountry' 'EE' 'SE' 'ZA' 'IS' 'BG' 'IR' 'CH']

Número de idiomas: 17 Entre los que se incluye nan

array(['english', 'german', 'french', 'spanish', 'russian', 'greek',
       'finnish', 'portuguese', 'arabic', 'dutch', 'italian', 'turkish',
       'ignore', 'norwegian', 'chinese', 'polish', nan], dtype=object)
```

Ilustración 26: Países e idiomas de las noticias

Para comprobar la distribución de estos valores en función del tipo de noticia, separamos las noticias verdaderas de las falsas. Tras esto descubrimos que las noticias reales solo están en inglés, también aparecen algunas con el campo “language” en nulo, pero al realizar un análisis superficial de los datos (y dado el hecho de que las fuentes de las noticias reales son diarios estadounidenses) se ha podido concluir que están en inglés.

En lo que respecta a las noticias falsas, 11.349 de las 11.941 noticias clasificadas como tal están en inglés (95%), por lo que las noticias en otros idiomas podrían considerarse como ruido que provocaría que noticias en un idioma distinto al inglés fueran clasificadas como falsas. La distribución de las noticias falsas por idioma es mostrada en la Ilustración 27.

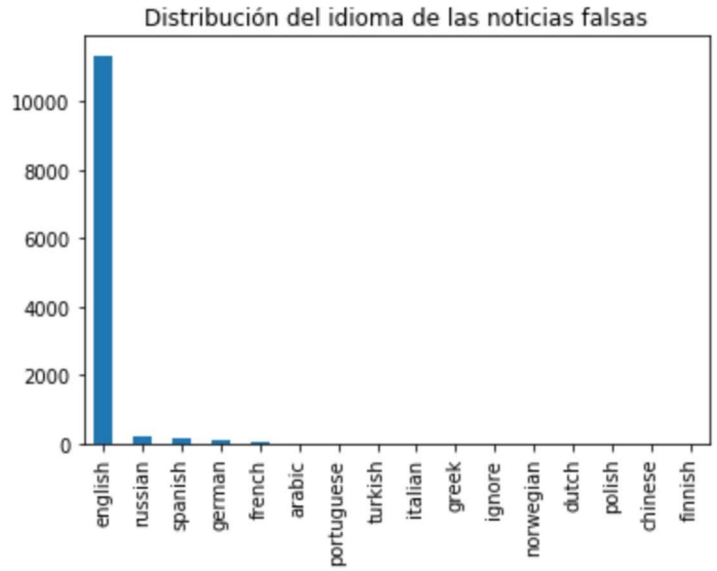


Ilustración 27: Distribución de las noticias falsas por idioma

Nos quedamos solamente con las noticias en inglés y nan, lo que nos deja con un total de 19.423 noticias.

```
df = df.loc[(df['language'] == 'english') | (df['language'].isna())]
```

Ilustración 28: Filtrado por idioma

En cuanto a la distribución por países, todas las noticias verdaderas son de Estados Unidos, el 39% de las noticias verdaderas no indican su procedencia, aunque dadas las fuentes podemos asumir que también lo son. La mayoría de las fake news también pertenecen a Estados Unidos, con Gran Bretaña en segundo lugar, como muestra la siguiente gráfica.

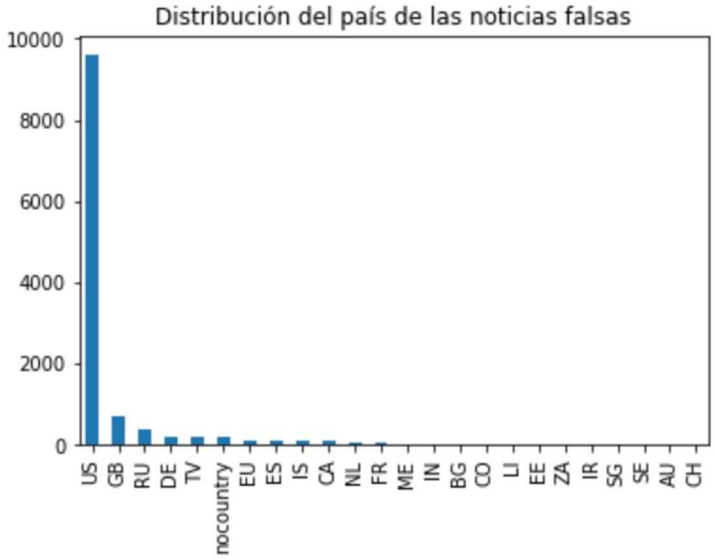


Ilustración 29: Distribución de las noticias falsas por país

6.3.1.2. Análisis de las fuentes

De acuerdo con el paper original [1], las noticias reales han sido extraídas de fuentes como el New York Times y el Washington Post, pero no se dan más detalles, por lo que puede resultar interesante estudiar dichas fuentes.

Si imprimimos el número de fuentes únicas obtenemos un total de 4093, no obstante, al observar la lista nos damos cuenta de que aparece la url completa de la noticia como fuente, por lo que no podemos considerar cada una de estas fuentes como únicas.

Mediante la ejecución del código mostrado en la Ilustración 30, eliminamos las rutas de las urls manteniendo solamente el dominio, teniendo en cuenta también aquellas entradas en las que la fuente no se indique (valor nulo).

```
fuentes = list(df_real.site_url.unique())

nan_count = df_real['site_url'].isna().sum()
print('De las ' + str(len(df_real)) + ' noticias reales ' + str(nan_count) + ' no tienen indicada una fuente\n')
# Las 4903 noticias restantes no indican la página de la fuente, sino la url completa, por lo que procedemos a eliminar
# las rutas de la url para quedarnos solo con el dominio

from urllib.parse import urlparse

fuentes_filtradas = {'nan': nan_count}

for i in fuentes:

    # Los valores nan son de tipo float
    if type(i) is str:
        o = urlparse(i)

        fuente = o.netloc

        if fuente not in fuentes_filtradas:
            fuentes_filtradas[fuente] = 1
        else:
            fuentes_filtradas[fuente] = fuentes_filtradas[fuente] + 1

print('Las noticias reales tienen un total de ' + str(len(fuentes_filtradas)-1) + ' fuentes conocidas\n')
print(fuentes_filtradas)
```

Ilustración 30: Filtrado de las fuentes

Las noticias reales tienen, por tanto, un total de cuatro fuentes conocidas (www.nytimes.com, www.wsj.com, www.americannews.com y www.politico.com). En un 39,3% de los casos la fuente no se indica. La distribución completa es mostrada en la Ilustración 31:

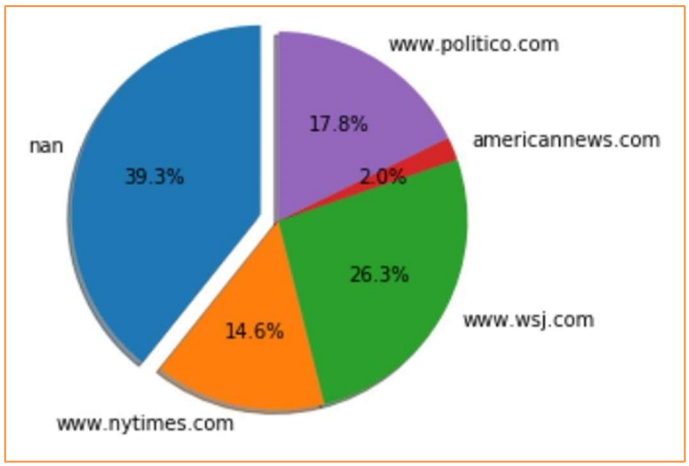


Ilustración 31: Fuentes de las noticias reales

6.3.1.3. Limpieza de datos y preprocesado

Podemos observar que, tras el filtrado por idioma, la proporción sigue manteniéndose casi idéntica.

Ilustración 32: Pie Chart tipos de noticias

Como se muestra en la Ilustración 32, el conjunto de datos no está balanceado, ya que el 58,4% de las noticias están etiquetadas como “fake”, mientras que el 41,6% restante está etiquetado como “real”. Cuando un dataset no está balanceado, suelen aplicarse técnicas de remuestreo (también conocido como resampling), existen dos posibles alternativas:

- **Oversampling:** Consiste en duplicar aleatoriamente entradas de la clase minoritaria. El principal inconveniente de este método es que puede provocar un sobreajuste, también conocido como Overfitting.
- **Undersampling:** Este método se basa en eliminar entradas aleatorias de la clase mayoritaria. Como inconveniente, la aplicación de esta técnica puede traer consigo cierta pérdida de información.

Dado el tamaño de los datos, aplicar Undersampling traería consigo una pérdida de información demasiado grande, mientras que el Overfitting provocado por la aplicación de Oversampling podría ser muy problemático. Por estas razones, y dado que el desequilibrio del dataset no es demasiado pronunciado, se ha optado por mantener esta proporción.

Según el paper de los creadores del dataset [1], la palabra más común en el caso de las fake news es “notitle”, que se emplea para sustituir al String vacío o a un valor NaN en el caso de que las noticias no tengan título. Como puede observarse en la Ilustración 33, esta palabra no aparece en ninguna noticia real.

```
count_notitle_fake = len(df.loc[(df['type'] == 'fake') & (df['title'] =='notitle')])
count_notitle_real = len(df.loc[(df['type'] == 'real') & (df['title'] =='notitle')])

print(count_notitle_real)
print(count_notitle_fake)
```

```
0
591
```

Ilustración 33: Número de Noticias con título nulo

Dejar los títulos nulos como “notitle” podría considerarse como una opción viable, pero creemos que la razón por la que las noticias falsas no tienen título no se debe a la estructura de las páginas, sino a la forma en la que se ha realizado el Web Scraping sobre las mismas. Definimos la función delete_notitle y la aplicamos a todo el dataset.

```
def delete_notitle(title):
    if title == 'notitle':
        return ' '
    else:
        return title
```

```
df['title'] = df['title'].apply(delete_notitle)
```

Ilustración 34: Aplicación de la función delete_notitle

Eliminamos todas las columnas exceptuando “title”, “text” y “type”. Unimos el título y el cuerpo de la noticia y eliminamos a columna “title” (Ilustración 35)

Unimos título y texto en una sola columna

```
df['text'] = df['title'] + ' ' + df['text']
```

Eliminamos la columna título

```
df = df.drop(columns = ['title'])
```

Ilustración 35: Eliminación de la columna title

Convertimos la variable type a One-hot-Encoding (valdrá 1 si la noticia es fake y 0 en caso contrario) para que pueda ser interpretada por los modelos neuronales.

```
df.loc[df['type'] == 'fake', 'type'] = 1.
df.loc[df['type'] == 'real', 'type'] = 0.
```

Ilustración 36: Conversión de la variable Type a One-Hot-Encoding

Por último, aplicamos la función tokenizer y exportamos el dataset preprocesado:

```
# Preprocesado
df['text'] = df['text'].apply(nlp_f.tokenizer)

# Exportar
df.to_csv('../dataset/ticnn_preprocessed.csv', index=False)
```

Ilustración 37: Preprocesar y Exportar dataset

Una vez preprocesado el conjunto de datos, calculamos la media, la desviación típica y la Noticia más larga de cada categoría en función del número de palabras. La fórmula empleada para el cálculo de la Desviación típica es la mostrada en la Ecuación 14:

$$\sigma = \sqrt{\frac{\sum(x - \bar{x})^2}{n - 1}}$$

Ecuación 14: Desviación típica

En base a los resultados de la Tabla 28 podemos observar que, en base a la media aritmética, la diferencia de longitud entre las noticias no es demasiado pronunciada, aunque la desviación típica indica que la media en el caso de las noticias falsas tiene una mayor dispersión.

	Media Aritmética	Desviación típica	Noticia más larga
Real	485,11	352,42	4.439
Fake	372,23	523,75	12.212
Total	419,11	463,67	12.212

Tabla 28: Estadísticas de la longitud de las noticias

6.3.2. Preprocesamiento

Las técnicas de preprocesamiento seleccionadas en la sección Diseño inicial se han combinado en la función mostrada en la Ilustración 38, que recibe el texto como parámetro y lo devuelve Preprocesado.

```
def tokenizer(example_sent):
    # Capitalization
    example_sent = str(example_sent).lower()
    # HTML TAGS
    example_sent = BeautifulSoup(example_sent, 'lxml').text
    # EMAIL ADDRESSES
    example_sent = re.sub(r'[\w\.-]+@[\w\.-]+', ' ', example_sent)
    # URLs
    example_sent = re.sub(r'http\S+', '', example_sent)
    # Punctuation
    tokenizer = RegexpTokenizer(r'\w+')
    word_tokens = tokenizer.tokenize(example_sent)
    # POS Tagging
    tags = nltk.pos_tag(word_tokens)
    # Lemmatization
    for i, word in enumerate(word_tokens):
        word_tokens[i] = lemmatizer.lemmatize(word, pos=pos_map.get(tags[i][1], 'n'))

    # stop words
    filtered_sentence = [w for w in word_tokens if not w in stop_words]
    # digits
    filtered_sentence = [w for w in filtered_sentence if not w.isdigit()]

    return filtered_sentence
```

Ilustración 38: Función Preprocesado

La Ilustración 39 muestra un ejemplo de Preprocesamiento paso a paso de una cadena de texto.

<HTML>This is.a ¡ sample@gmail.com sentences, showing off the <br> stop words filtration. http://www.youtube.com

CAPITALIZATION

<html>this is.a ¡ sample@gmail.com sentences, showing off the <br> stop words filtration. http://www.youtube.com

HTML TAGS

this is.a ¡ sample@gmail.com sentences, showing off the stop words filtration. http://www.youtube.com

EMAILS Y URL

this is.a ¡ sentences, showing off the stop words filtration.

SIGNOS DE PUNTUACIÓN

[this, is, a, sentences, showing, off, the stop, words, filtration]

POS TAGGING

[(this, DT), (is, VBZ), (a, DT), (sentences, NNS), (showing, VBG), (off, RP), (the, DT), (stop, NN), (words, NNS), (filtration, NN)]

LEMMATIZATION

[this, is, a , sentence, show, off, the, stop, word, filtration]

STOP WORDS

[sentence, show, stop, word, filtration]

Ilustración 39: Ejemplo Preprocesamiento

6.4. Pruebas

Las pruebas en este sprint se han centrado en comprobar el correcto funcionamiento de la lectura del conjunto de datos y su preprocesado, con la intención de detectar posibles errores en la implementación.

Gracias a estas pruebas se ha descubierto que la versión inicial de la función de preprocesado era incapaz de preprocesar strings vacíos, lo que provocaba un error a la hora de aplicar la función al conjunto de datos completo.

P-01: Lectura del fichero CSV	
Requisitos	REQ-01
Descripción	Lectura de un subconjunto del fichero csv
Pre-requisitos	Tener guardado el fichero subset_test.csv en la misma carpeta que el notebook de Jupyter en el que se realizan las pruebas
Entrada	Fichero csv
Salida esperada	Objeto Dataframe de pandas representando la información contenida en el csv
Salida obtenida	Objeto Dataframe de pandas representando la información contenida en el csv

Tabla 29: Prueba 1 - Lectura del fichero csv

El código de esta prueba se muestra en la Ilustración 40 y los resultados aparecen en la Ilustración 41.

Prueba 1: Lectura del fichero csv

In [3]:

```
def prueba1():
    print("Lectura del fichero csv. \n")
    print("El archivo subset_test.csv debe encontarse en la misma carpeta que este notebook")

    try:
        datos = pd.read_csv('subset_test.csv')
    except:
        print('Error al leer el fichero csv')
        raise

    print("El fichero se ha leído correctamente. Los primeros cinco elementos son:")
    print(datos.head())
    print('-'*40)
    print('Los últimos 5 elementos son:')
    print(datos.tail())
```

Ilustración 40: Prueba 1 - Lectura del Fichero csv

```
In [4]: prueba1()

Lectura del fichero csv.

El archivo subset_test.csv debe encontarse en la misma carpeta que este notebook
El fichero se ha leído correctamente. Los primeros cinco elementos son:
   type                                              title  \
0   1.0  EPA Delays Rule to Limit Carbon Emissions from...
1   1.0  Marco Rubio Signals Possible Opposition To Rex...
2   1.0  UPDATE: Boehner Power Point of deal added- Sun...
3   1.0  Is Lack of Solemnity a Cause or a Symptom of O...
4   1.0  Free 45 minute session with “PlayBook pro spec...

                                             content
0  EPA Delays Rule to Limit Carbon Emissions from...
1  Marco Rubio Signals Possible Opposition To Rex...
2  UPDATE: Boehner sent a Power Point to his cauc...
3  (Before It's News)\n\nThe Ordinary Form at the...
4  Free 45 minute session with “PlayBook pro spec...
----------------------------------------
Los últimos 5 elementos son:
      type                                              title  \
3995   0.0                                            Wolfish
3996   0.0  Castro's Cuba, beacon in Latin America's lefti...
3997   0.0  Americas Silver Corporation Reports Third Quar...
3998   0.0    WATCH: Catching Up With the Cast of 'Moonlight'
3999   0.0          How Trade Deals Have Hurt American Workers
```

Ilustración 41: Prueba 1 - Lectura del fichero csv (Resultados)

P-02: Preprocesado del dataset	
Requisitos	REQ-02
Descripción	Aplicar la función tokenizer a un subconjunto del dataset
Pre-requisitos	Tener guardado el fichero subset_test.csv en la misma carpeta que el notebook de Jupyter en el que se realizan las pruebas y el archivo nlp_functions.py en el directorio raíz del proyecto.
Entrada	Fichero csv
Salida esperada	Objeto Dataframe representando el dataset preprocesado
Salida obtenida	Objeto Dataframe representando el dataset preprocesado

Tabla 30: Prueba 2 - Preprocesado del dataset

El código de esta prueba se muestra en la Ilustración 42 y los resultados aparecen en la Ilustración 43

Prueba 2: Preprocesado del dataset

In [6]:
```
def prueba2():
    try:
        print("Lectura del fichero... \n")
        datos = pd.read_csv('subset_test.csv')
    except:
        print("Error al leer el fichero csv")
        raise
    print("El fichero se ha leído correctamente")

    print("Primeras tres entradas sin preprocesar")
    print(datos.head(3))
    print('-'*40)

    try:
        print("Preprocesado del dataset")
        datos['content'] = datos['content'].apply(nlp_f.tokenizer)
    except:
        print("Error al preprocesar el conjunto de datos")
        raise
    print("El preprocesado se ha realizado correctamente")
    print("Primeras tres entradas preprocesadas")
    print(datos.head(3))
```

Ilustración 42: Prueba 2 - Preprocesado del dataset

```
In [9]: prueba2()

        Lectura del fichero...

        El fichero se ha leído correctamente
        Primeras tres entradas sin preprocesar
           type                                              title  \
        0   1.0  EPA Delays Rule to Limit Carbon Emissions from...
        1   1.0  Marco Rubio Signals Possible Opposition To Rex...
        2   1.0  UPDATE: Boehner Power Point of deal added- Sun...

                                                     content
        0  EPA Delays Rule to Limit Carbon Emissions from...
        1  Marco Rubio Signals Possible Opposition To Rex...
        2  UPDATE: Boehner sent a Power Point to his cauc...
        ----------------------------------------
        Preprocesado del dataset
        El preprocesado se ha realizado correctamente
        Primeras tres entradas preprocesadas
           type                                              title  \
        0   1.0  EPA Delays Rule to Limit Carbon Emissions from...
        1   1.0  Marco Rubio Signals Possible Opposition To Rex...
        2   1.0  UPDATE: Boehner Power Point of deal added- Sun...

                                                     content
        0  [epa, delay, rule, limit, carbon, emission, ne...
        1  [marco, rubio, signal, possible, opposition, r...
        2  [update, boehner, send, power, point, caucus, ...
```

Ilustración 43: Prueba 2 - Preprocesado del dataset (Resultados)

P-03: Preprocesado de un objeto que no es de tipo string	
Requisitos	REQ-02
Descripción	Preprocesar un objeto que no es de tipo string utilizando la función tokenizer, esta prueba es necesaria para evitar que la ejecución del preprocesado del dataset completo se detenga por la existencia de ruido en el conjunto de datos
Pre-requisitos	Tener guardado el fichero nlp_functions.py en el directorio raíz del proyecto.
Entrada	Un objeto de tipo float
Salida esperada	Una lista vacía
Salida obtenida	Una lista vacía

Tabla 31: Prueba 3 - Preprocesado de un objeto que no es de tipo String

El código y los resultados de esta prueba son los mostrados en la Ilustración 44

Prueba 3: Preprocesado de un objeto que no es de tipo string

```
In [10]: def prueba3():

             test = 123456.6

             try:
                 print('Preprocesado de un objeto que no es de tipo String')
                 test = nlp_f.tokenizer(test)
             except:
                 print("Error al preprocesar una entrada que no es de tipo String")
                 raise

             print('El preprocesado se ha realizado correctamente')
             print(test)

In [11]: prueba3()

         Preprocesado de un objeto que no es de tipo String
         El preprocesado se ha realizado correctamente
         []
```

Ilustración 44: Prueba 3 - Preprocesado de un objeto que no es de tipo string

P-04: Preprocesado de un String	
Requisitos	REQ-02
Descripción	Preprocesar un string que contenga todos los casos de preprocesado definidos en el diseño de dicha función
Pre-requisitos	Tener guardado el fichero nlp_functions.py en el directorio raíz del proyecto.
Entrada	Un objeto de tipo string
Salida esperada	Una lista de strings con el preprocesado realizado correctamente
Salida obtenida	Una lista de strings con el preprocesado realizado correctamente

Tabla 32: Prueba 4 - Preprocesado de un String aplicando todas las técnicas de preprocesado

El código y los resultados de esta prueba aparecen en la Ilustración 45

Prueba 4: Preprocesado de un string aplicando todas las técnicas de preprocesado

```
In [12]: def prueba4():
             test = "<HTML>This <p>is.a</p> ! jua@email.com sentences, showing off the <br> stop words filtration. http://www.youtube.com

             try:
                 print('Preprocesado de un string')
                 test = nlp_f.tokenizer(test)
             except:
                 print('Error al preprocesar el string')
                 raise
             print('El preprocesado se ha realizado correctamente')
             print(test)

In [13]: prueba4()

         Preprocesado de un string
         El preprocesado se ha realizado correctamente
         ['sentence', 'show', 'stop', 'word', 'filtration']
```

Ilustración 45: Prueba 4 - Preprocesado de un string aplicando todas las técnicas de preprocesado

6.5. Resumen del Sprint

El desarrollo de este primer sprint constituye una parte muy importante para la consecución de los objetivos del proyecto, ya que un buen procesamiento y análisis de los datos resulta vital para el correcto funcionamiento de los modelos de clasificación que se implementarán en futuros Sprints.

Durante este sprint también han cobrado gran importancia las labores de documentación referentes a la planificación y diseño inicial, entre otras. Todos los objetivos del Sprint backlog se han cumplido dentro de los plazos establecidos, sin sufrir ningún retraso.

Las horas trabajadas han sido cercanas a las estimadas para este sprint (40 horas), si bien es cierto que se ha dedicado un poco más de tiempo a labores de documentación de lo que se estableció en un inicio. Afortunadamente, el tiempo empleado para tareas de implementación ha acabado siendo inferior al estimado. Se han dedicado un total de 8 horas a labores de investigación.

En la Ilustración 46 se muestra el gráfico Burndown de este primer sprint, que refleja que se ha seguido un ritmo de trabajo constante y continuo. Puede observarse claramente que el valor real se sitúa por debajo del valor ideal en todo momento, lo cual muestra que no ha habido ningún retraso en ninguna de las tareas a realizar.

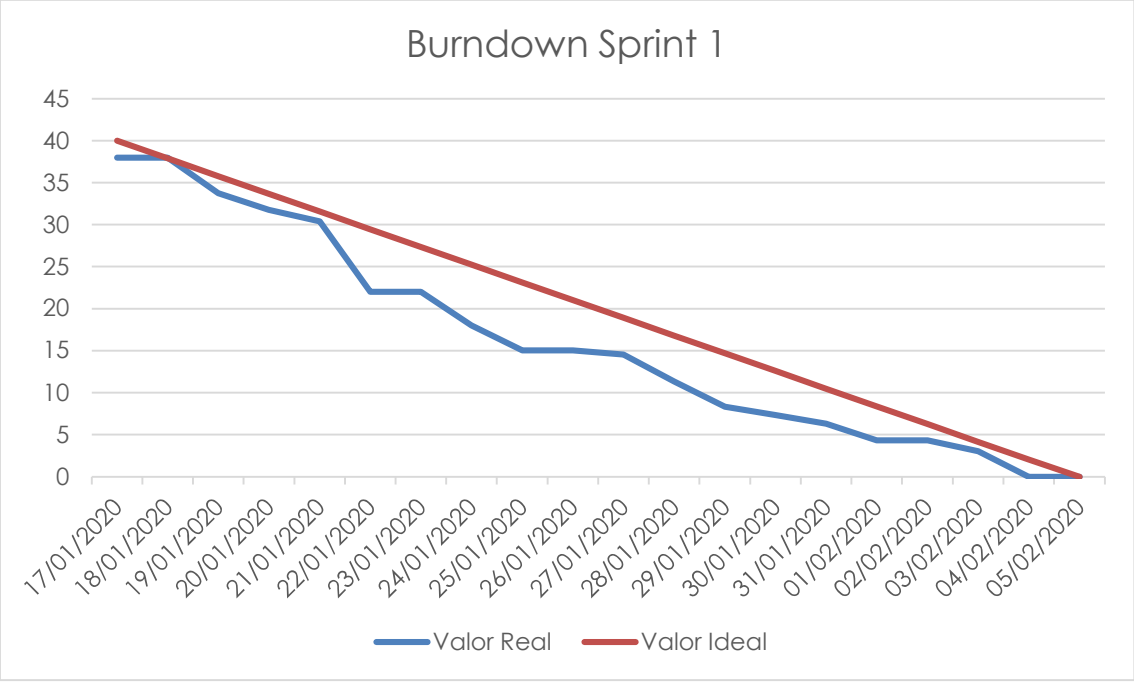


Ilustración 46: Burndown Sprint 1

7. Sprint 2: Aplicación de baselines

A lo largo de este sprint, se aplicarán los primeros modelos de clasificación sobre el conjunto de datos. En Machine Learning, el término baseline se emplea para referirse a un modelo simple que se aplica sobre el conjunto de datos antes de emplear modelos más complejos. Suelen consistir en aplicar técnicas estadísticas sobre los datos. La aplicación de baselines cumple diversas funciones: nos permiten familiarizarnos con el conjunto de datos (un análisis exhaustivo del conjunto de datos de tamaño medio no puede ser realizado a mano por un ser humano), detectar posibles sesgos y tener un punto de partida con el que comparar los modelos de mayor complejidad que se implementarán posteriormente [60].

A pesar del nombre que recibe este sprint, no todas las técnicas que van a aplicarse durante el transcurso de este son baselines, las relativas a Word2Vec y Doc2Vec son bastante avanzadas y son conocidas por dar muy buenos resultados en la práctica.

Para cada uno de los métodos, se mostrarán las ventajas e inconvenientes, y se compararán con métodos similares, ya que en Data Science nunca se puede afirmar que un modelo sea mejor que otro, cada método tiene sus usos y puede ser más adecuado para una determinada tarea que otros. Este sprint tiene una duración de 26 días, con un total de 60 horas asignadas.

7.1. Requisitos

Los requisitos que se abordarán en este sprint son los siguientes:

REQ-03: Bag of Words – Binary Counts
Aplicar Binary Words usando Binary Counts para la clasificación de los datos, siguiendo los principios de elección, entrenamiento y evaluación del modelo.

Tabla 33: REQ-03

REQ-04: Bag of Words – Term Frequency
Aplicar Binary Words usando Term Frequency para la clasificación de los datos, siguiendo los principios de elección, entrenamiento y evaluación del modelo.

Tabla 34: REQ-04

REQ-05: Bag of Words – Tf-idf
Aplicar Binary Words usando Term Frequency e Inverse Document Frequency para la clasificación de los datos, siguiendo los principios de elección, entrenamiento y evaluación del modelo.

Tabla 35: REQ-05

REQ-06: Word2Vec – Skipgram
Aplicar Word2Vec usando Skipgram para la clasificación de los datos, siguiendo los principios de elección, entrenamiento y evaluación del modelo.

Tabla 36: REQ-06

REQ-07: Word2Vec – CBOW
Aplicar Word2Vec usando Continuous Bag Of Words (CBOW) para la clasificación de los datos, siguiendo los principios de elección, entrenamiento y evaluación del modelo.

Tabla 37: REQ-07

REQ-08: Doc2Vec – DM
Aplicar Doc2Vec usando Distributed Memory (DM) para la clasificación de los datos, siguiendo los principios de elección, entrenamiento y evaluación del modelo.

Tabla 38: REQ-08

REQ-09: Doc2Vec – DBOW
Aplicar Doc2Vec usando Distributed Bag Of Words (DBOW) para la clasificación de los datos, siguiendo los principios de elección, entrenamiento y evaluación del modelo.

Tabla 39: REQ-09

7.2. Diseño

7.2.1. Diagrama de clases

El diagrama de clases a implementar en este Sprint es el mostrado en la Ilustración 47, este diagrama se centra en las clases model_creation, model_training y model_evaluation, en las que se va a trabajar durante este sprint y el siguiente (la lectura y el preprocesado del dataset son los implementados a lo largo del Sprint 1).

Como sus nombres indican, las clases model_creation, model_training y model_evaluation se encargan de la creación, entrenamiento y evaluación del modelo respectivamente.

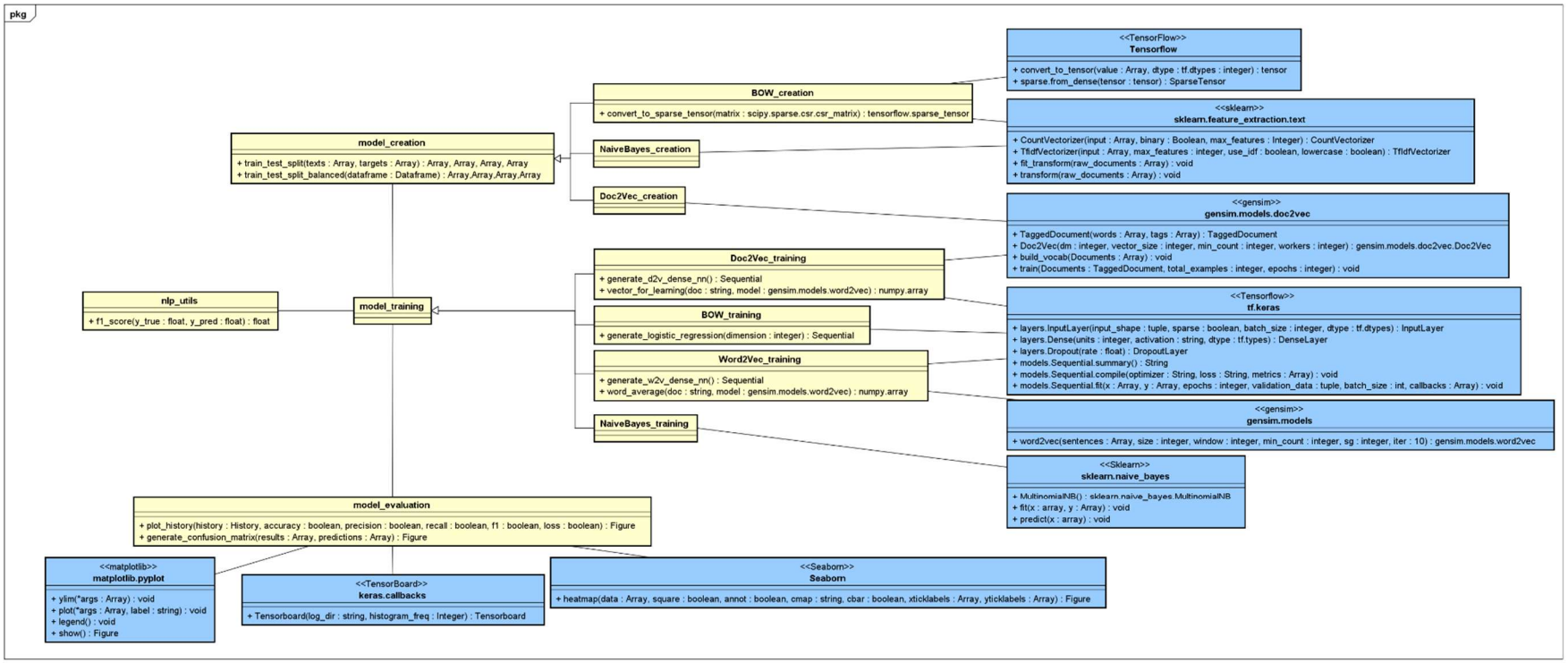


Ilustración 47: Diagrama de Clases a implementar en el Sprint 2

7.2.2. Modelos basados en Bag of Words

El modelo Bag of Words (BOW) es una representación simplificada de documentos utilizada en procesamiento del lenguaje natural y en extracción de información [61]. En este modelo, un texto es representado como un multiset de las palabras que lo componen, sin tener en cuenta la gramática ni el orden de las palabras, pero manteniendo la información referente a la multiplicidad.

Los modelos basados en Bag of Words suelen emplearse en tareas relativas a clasificación de documentos, en las que la frecuencia de cada palabra se utiliza como dato entrenar un clasificador. También puede utilizarse como medida de similitud entre documentos utilizando métricas como la Similitud del coseno o la distancia Euclídea.

A pesar de que los modelos basados en Bag of Words suelen ofrecer resultados muy buenos en la práctica con datasets pequeños, estos tienen una serie de desventajas que es necesario tener en cuenta [62]:

- El vocabulario debe ser diseñado adecuadamente, prestando especial atención a su tamaño, que afecta a lo dispersos que sean los vectores y al almacenamiento.
- Las representaciones vectoriales dispersas son difíciles de modelar por razones computacionales (complejidad temporal y espacial) y también por razones referentes a información, ya que se almacena muy poca información empleando una gran cantidad de espacio.
- Al ignorar el orden de las palabras, se ignoran la semántica y muchos aspectos del contexto, que pueden ofrecer mucha información al modelo. Además de esto, al representar los documentos como vectores dispersos, no existe ninguna noción de similitud entre palabras que nos permitan inferir relaciones como la sinonimia o la antonimia entre términos.

Se implementarán tres versiones distintas del modelo Bag of Words, Binary Term Occurrence, Term Frequency y Term Frequency - Inverse Document Frequency. La estructura de estos modelos es muy similar, las diferencias de implementación entre las distintas variantes se explicarán en sus respectivas secciones dedicadas a la implementación.

Una vez obtenidas las representaciones vectoriales de cada uno de los documentos en el formato correspondiente, se aplicará regresión logística para su clasificación. Para la representación gráfica de los resultados, se emplearán gráficas con cada una de las métricas definidas en la sección Métricas utilizadas.

7.2.2.1. Binary Term Occurrences (Apariciones Binarias de términos)

Las apariciones binarias son la versión más simple de BOW. Se utilizan atributos booleanos (1 y 0 en la implementación para poder realizar los cálculos) para representar la aparición o ausencia de un término en un documento [63]. A continuación, se muestra un ejemplo de una matriz de Bag of Words con apariciones binarias para el siguiente corpus compuesto por tres documentos:

D1: "Frodo stabbed the orc with the red sword"

D2: "Frodo and Sam used the blue lamp to locate orcs"

D3: "Sam killed many orcs in Mordor with the blue sword"

Tras aplicar lematización y eliminar las stopwords, el vocabulario del corpus sería:

$$V = \{Frodo, Sam, blue, sword, orc, Mordor, red, stab, lamp, locate, kill\}$$

Ecuación 15: Vocabulario de ejemplo

Esta información puede expresarse como una matriz de incidencia como la de la Tabla 40:

	Frodo	Sam	Blue	Sword	Orc	Mordor	Red	Stab	Lamp	Locate	Kill
D1	1	0	0	1	1	0	1	1	0	0	0
D2	1	1	1	0	1	0	0	0	1	1	0
D3	0	1	1	1	1	1	0	0	0	0	1

Tabla 40: Matriz de Incidencia Binaria

Cada una de las filas de la tabla anterior es la representación vectorial del documento correspondiente, con un tamaño fijo igual al tamaño del vocabulario. El tamaño de los vectores es uno de los mayores problemas de los modelos basados en BOW, ya que almacenar vectores de este tamaño suele traer consigo problemas de memoria con datasets de tamaño medio. Además de esto, se trata de vectores dispersos (la mayoría de los valores son 0) que no permiten comparar la similitud entre términos.

El principal problema de la versión basada en apariciones binarias es que no se tiene en cuenta la rareza de cada palabra, solo si aparece o no en el documento. Por ejemplo, en la matriz de incidencia de la Tabla 40, la aparición de la palabra "orc" tiene el mismo valor que la palabra "lamp", cuando la palabra "orc" aparece en todos los documentos (por lo que podemos considerar que su aparición nos aporta poca información), mientras que la palabra "lamp" solamente aparece en el documento 2.

7.2.2.2. Term Frequency (Frecuencia de Términos)

Si utilizamos la frecuencia de cada término para rellenar la matriz de incidencia, los valores de la matriz representan la frecuencia con la que cada término aparece en cada documento de acuerdo con la Ecuación 16 [63]:

$$tf_{t,d} = \begin{cases} 1 + log_{10}(count(t,d)), & Si\ count(t,d) > 0 \\ 0, & En\ otro\ caso \end{cases}$$

Ecuación 16: Cálculo de Term Frequency

El logaritmo se emplea para que la fórmula no sea lineal. De esta forma, no solo se tiene en cuenta si la palabra aparece o no en un documento, sino también su frecuencia, lo que nos permite inferir si una palabra tiene más o menos importancia para el documento en cuestión.

El problema de este método es que sigue sin tener en cuenta la frecuencia de los términos en el resto de los documentos, además de seguir teniendo los problemas de almacenamiento mencionados previamente. A pesar de esto, supone una mejora considerable con respecto a Apariciones binarias.

7.2.2.3. TF-IDF (Term Frequency – Inverse Document Frequency)

Term Frequency – Inverse Document Frequency (generalmente abreviado como tf-idf) es la variante más avanzada de la técnica del Bag of Words, y generalmente es la que suele dar mejores resultados a la hora de clasificar documentos [64].

El problema con utilizar la frecuencia de las palabras para generar la matriz de incidencia es que las palabras que aparezcan con mucha frecuencia tienden a dominar en el documento, pero no suelen contener información tan valiosa para el modelo como ciertas palabras específicas del dominio [62] [65] tf-idf se utiliza para evaluar como de importante es una palabra para un corpus de documentos, dando más peso a palabras con un menor número de apariciones, y menor peso a palabras más comunes. El valor de tf-idf para cada término t de un documento d se calcula como el producto de la frecuencia del término t en el documento d multiplicada por la inversa de la frecuencia documental del término t.

$$w_{t,d} = tf_{t,d} \cdot idf_t$$

Ecuación 17: Cálculo de tf-idf

La fórmula para obtener la frecuencia del término es la indicada en la Ecuación 17, mientras que la frecuencia documental inversa de un término t se calcula de la siguiente forma:

$$idf_t = \log_{10}\left(\frac{N}{df_t}\right)$$

Ecuación 18: Cálculo de idf

Donde N es el número total de documentos en el corpus y df_t es el número de documentos en los que aparece el término t, de esta forma obtenemos una métrica que da mayor valor a los términos si aparecen en un número menor de documentos. La matriz de incidencia empleando tf-idf quedaría como se muestra en la Tabla 41:

	Frodo	Sam	Blue	Sword	Orc	Mordor	Red	Stab	Lamp	Locate	Kill
D1	0,176	0	0	0,176	**0**	0	**0,47**	**0,47**	0	0	0
D2	0,176	0,176	0,176	0	**0**	0	0	0	**0,47**	**0,47**	0
D3	0	0,176	0,17	0,176	**0**	**0,47**	0	0	0	0	**0,47**

Tabla 41: Matriz de Incidencia tf-idf

Comparando esta matriz de incidencia con la generada por apariciones binarias, podemos notar dos claras diferencias, que han sido marcadas en rojo. En primer lugar, la palabra “orc”, que aparece en todos los documentos pasa a tener valor cero, ya que aporta poca o ninguna

información que nos permita distinguir a los textos. En cambio, palabras como “lamp”, que solo aparecen en un documento, son las que tienen asignado un valor mayor en la matriz de incidencia, ya que nos permiten encontrar diferencias entre los distintos documentos.

7.2.3. Naive Bayes

La clasificación con Naive Bayes está basada en el teorema de Bayes, que enuncia que la probabilidad de que se cumpla una hipótesis dadas ciertas evidencias es igual a la probabilidad de la hipótesis multiplicada por la probabilidad de la evidencia dada la hipótesis, y dividido por la probabilidad de la evidencia [66]:

$$P(y|x) = \frac{P(x|y) \cdot P(y)}{P(x)}$$

Ecuación 19: Teorema de Bayes

En la Ecuación 19, y es la hipótesis o, en el caso de que el método se esté aplicando para clasificación, es una de las distintas clases en las que puede clasificarse una instancia, mientras que x es una de las propiedades de la instancia a clasificar. Por lo tanto, los distintos operadores de la fórmula anterior representan:

- $P(y|x)$: Probabilidad de que la instancia pertenezca a la clase y sabiendo que posee la propiedad x.
- $P(y)$: Probabilidad de que la instancia pertenezca a la clase y (Cociente entre el número de entradas del dataset que pertenecen a la clase y con el número total de entradas).
- $P(x|y)$: Probabilidad de que, sabiendo que una instancia pertenece a la clase y, posea la propiedad x.
- $P(x)$: Probabilidad de que una instancia posea la propiedad x.

La fórmula del teorema de Bayes puede extenderse con más variables, como se muestra en la Ecuación 20:

$$P(y|x_1, x_2, \dots, x_n) = \frac{P(x_1, x_2, \dots, x_n|y) \cdot P(y)}{P(x_1, x_2, \dots, x_n)}$$

Ecuación 20: Teorema de Bayes multivariable

Donde cada una de las variables del conjunto de x es una de las palabras del documento, mientras que las distintas clases a las que puede pertenecer el documento son los posibles valores de y. Por lo tanto, lo que se busca es, dada una secuencia de palabras $x_1, x_2, \dots, x_n$ que componen un documento, encontrar la clase que maximiza el valor de $P(Y|x_1, x_2, \dots, x_n)$. Esto es equivalente a:

$$P(Y|x_1, x_2, \dots, x_n) = P(x_1, x_2, \dots, x_n|Y) \cdot P(Y)$$

Ecuación 21: Teorema de Bayes sin denominador

Podemos omitir el denominador $P(x_1, x_2, \dots, x_n)$ porque es común a todas las clases, por lo que la proporción se mantiene.

Para calcular $P(x_1, x_2, \dots, x_n|y)$ asumimos que las variables del conjunto x son independientes entre sí, obteniendo la siguiente expresión:

$$\hat{y} = \underset{y}{\operatorname{argmax}} P(y) \prod_{i=1}^{n} P(x_i|y)$$

Ecuación 22: Naive Bayes

A pesar de su aparente simplicidad, los clasificadores basados en Naive Bayes devuelven muy buenos resultados en situaciones del mundo real de cierta complejidad relativas a clasificación de documentos, como filtros de spam, dando buenos resultados incluso con pocos datos. Se trata de un modelo relativamente fácil de entrenar (no hay ajuste de hiperparámetros más allá del tamaño del vocabulario) y es multiclase por defecto. No obstante, al asumir independencia entre variables, no es lo suficientemente complejo como para encontrar relaciones entre atributos [67].

Al no tratarse de un modelo entrenado por épocas, para la representación gráfica de los resultados durante la fase de Evaluación del modelo se realizará mediante una matriz de confusión, utilizando para ello la librería Seaborn.

7.2.4. Word2Vec

Los modelos basados en Word2Vec fueron propuestos en 2013 por Mikolov et al. [68] y desde entonces han alcanzado una gran popularidad por los buenos resultados que ofrecen. Estos métodos representan a las palabras que constituyen el vocabulario de un corpus como vectores densos, a diferencia de los vectores dispersos que utilizan los modelos basados en Bag of Words.

Los vectores densos se basan en la hipótesis distribucional para obtener información sobre los términos que constituyen el vocabulario. La hipótesis distribucional sostiene que se puede inferir el significado de una palabra por su contexto. Las representaciones densas se derivan a partir de coocurrencias de palabras.

El número de dimensiones se reduce considerablemente en comparación con los modelos de Bag of Words, esta reducción de dimensiones mitiga los problemas de almacenamiento que existían en Bag Of Words, al concentrar una cantidad mayor de información en bastante menos espacio. El número de dimensiones puede seleccionarse, pero el valor estándar suele estar en 300 dimensiones para evitar una posible pérdida de información. También se introduce el concepto de distancia entre palabras, teniendo en cuenta no solo la aparición de estas en un documento, sino también su contexto y su semántica, además de relaciones sintácticas. Si el modelo se entrena adecuadamente, las palabras similares se encontrarán cerca en el espacio. Aunque las representaciones vectoriales ocupan menos memoria, generalmente se requieren más datos para realizar el entrenamiento.

La representación de las palabras en el espacio va más allá de simples relaciones sintácticas [68] ya que, realizando operaciones algebraicas simples sobre los vectores, puede obtenerse información muy valiosa. Por ejemplo, vector(“King”) – vector(“Man”) + Vector(“Woman”) devuelve un vector que está en una posición cercana al vector que representa a la palabra “Queen”. Otro ejemplo muy común es el de los países y las ciudades, que se muestra en la siguiente imagen [69]:

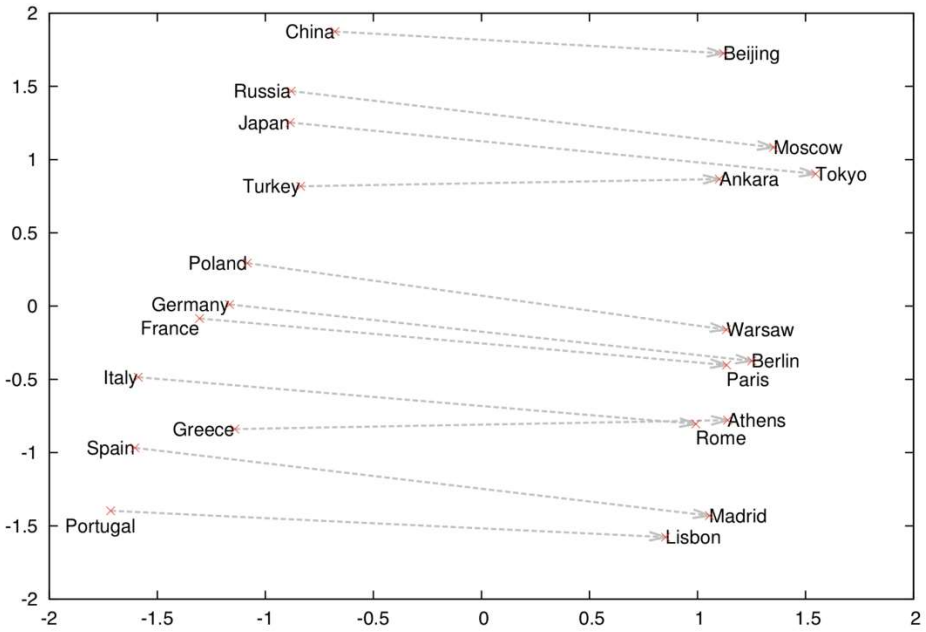


Ilustración 48: Proyecciones vectoriales de países y capitales

Los dos modelos existentes de Word2Vec son SkipGram y Continuous Bag of Words (CBOW). La arquitectura CBOW predice la palabra central basándose en el contexto, mientras que SkipGram predice el contexto a partir de la palabra central.

7.2.4.1. SkipGram

Skipgram permite inferir el contexto a partir de una palabra central. Una vez entrenado, el modelo SkipGram puede recibir la representación vectorial de una palabra, w(t), e inferir el contexto de esta. En este caso, con contexto nos referimos a las k palabras anteriores y posteriores (Ilustración 49).

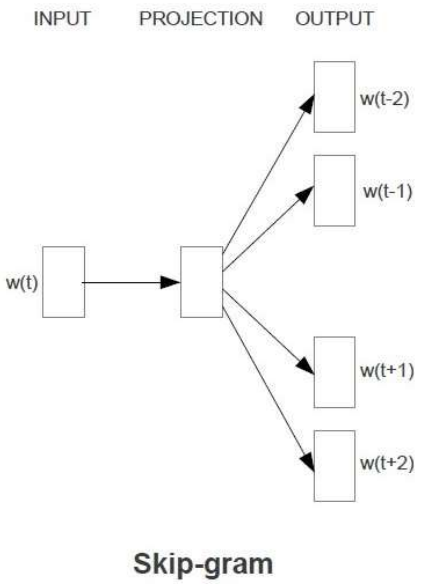


Ilustración 49: Modelo Skipgram [68]

Para entrenar un modelo Skipgram, se empieza asignando dos vectores densos aleatorios a cada palabra. Uno será el vector central y otro el del contexto. Para el entrenamiento se emplea una red neuronal de una sola capa en la que la entrada es una representación en One-Hot-Encoding de la palabra central. Las dos matrices son:

- $\boldsymbol{W}$ (Dimensión V x d, donde V es el tamaño del vocabulario y d es el número de dimensiones de los vectores densos): Transforma el vector en One-Hot-Encoding de la palabra central en un vector denso.
- $\boldsymbol{W'}$ (Dimensión d x V): Transforma el vector denso en el vector disperso del contexto.

Esta estructura es la mostrada en la Ilustración 50: [70]

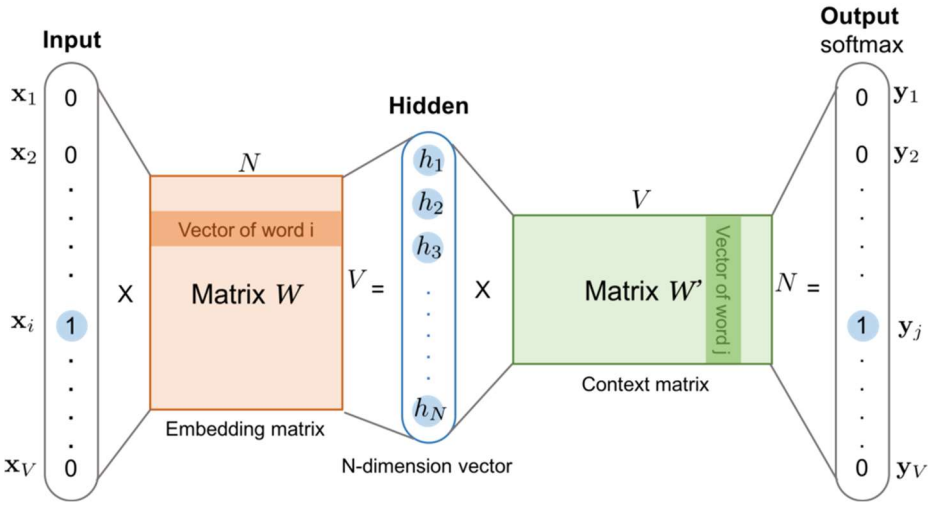


Ilustración 50: Estructura Neuronal de Skipgram

El vector denso central (de dimensión d) se calcula de acuerdo con la Ecuación 23:

$$c_w = v_w \cdot W$$

Ecuación 23: Cálculo del vector denso central

Donde v_w es el vector disperso de la palabra central. La predicción del contexto se calcula siguiendo la Ecuación 24:

$$c_p = c_w^T \cdot W'$$

Ecuación 24: Cálculo del contexto

El vector disperso del contexto no es una distribución de probabilidad, por lo que se le aplica la función softmax (Ecuación 25):

$$softmax(c_p^i) = \frac{e^{c_p^i}}{\sum_j e^{c_p^j}}$$

Ecuación 25: Función Softmax

Tras aplicar la función softmax, la representación de la palabra en la posición i se obtiene al concatenar:

1. La fila número i de la matriz W
2. La columna número i de la matriz W'

7.2.4.2. Continuous Bag of Words

Al contrario de lo que ocurre con Skipgram, la arquitectura CBOW predice la palabra central a partir del contexto (Ilustración 51).

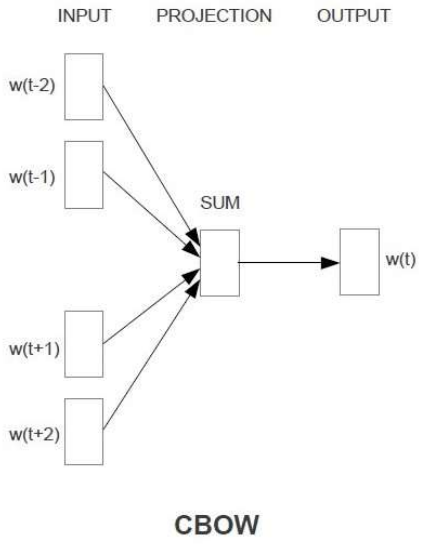


Ilustración 51: Modelo CBOW [68]

Al igual que en Skipgram, se comienza con dos vectores densos aleatorios para cada palabra (Vector central y vector del contexto, ambos de dimensión d).

De nuevo se emplea una red neuronal de una sola capa para el entrenamiento, salvo que en este caso la entrada es una representación en One-Hot-Encoding del contexto. Las dos matrices son:

- $\boldsymbol{W}$(Dimensión V x d): Transforma el vector en One-Hot-Encoding del contexto en un vector denso que lo representa.
- $\boldsymbol{W}'$ (Dimensión d x V): Transforma el vector denso del contexto en un vector disperso de la palabra central.

Esta estructura es la mostrada en la Ilustración 52 [70]:

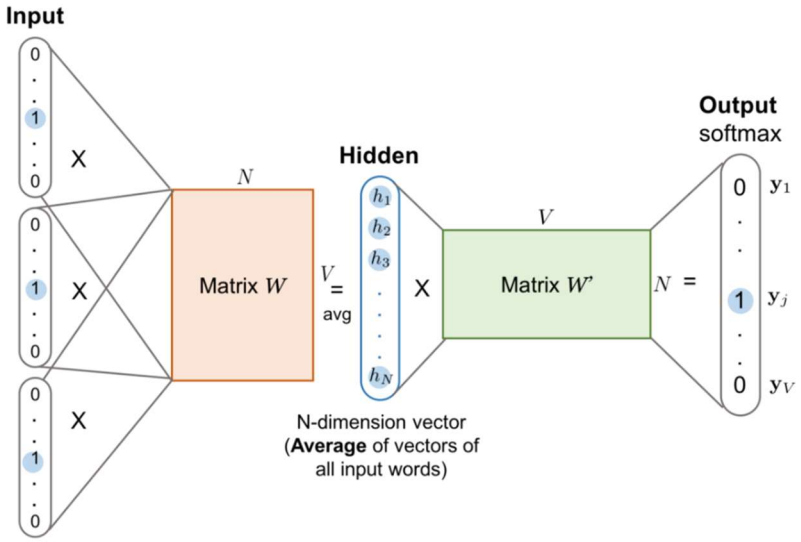


Ilustración 52: Estructura Neuronal de CBOW

Por último, se aplica la función softmax para obtener una distribución de probabilidad. Cuando el entrenamiento finaliza, la representación de la palabra en la posición i se obtiene concatenando:

1. La fila número i de la matriz W
2. La columna número i de la matriz W'

7.2.5. Doc2Vec

Los modelos basados en Doc2Vec fueron propuestos por Tomas Mikolov y Quoc Lec en 2014 [71] y son una versión extendida de Word2Vec que, en lugar de obtener una representación vectorial de cada palabra, obtiene una representación vectorial de cada documento. Estos modelos tienen la ventaja de tener en cuenta el orden de las palabras en el documento y la semántica de estas, propiedades que no se tienen en cuenta en la mayoría de las representaciones de documentos (como es el caso de Bag of Words).

Doc2Vec representa cada documento como un vector denso de longitud fija. En clasificación de textos, esta característica tiene dos ventajas principales:

- La clasificación no se verá sesgada por la variación de longitud entre documentos.
- Permite tener un input de tamaño fijo para estructuras como redes neuronales, sin tener que recortar ni paddear las secuencias.

Tanto los vectores de las palabras como los de los documentos son entrenados utilizando el descenso estocástico por el gradiente y el algoritmo de retropropagación. Las representaciones vectoriales de las palabras son compartidas (no son únicas de cada documento).

Existen dos versiones de Doc2Vec:

- Distributed Memory
- Distributed Bag of Words

7.2.5.1. Distributed Memory

Distributed Memory es una extensión del modelo CBOW de Word2Vec. En lugar de utilizar las representaciones vectoriales de las palabras para predecir otras palabras, extendemos el modelo añadiendo otro vector (Paragraph id), que es exclusivo de cada documento (Ilustración 53).

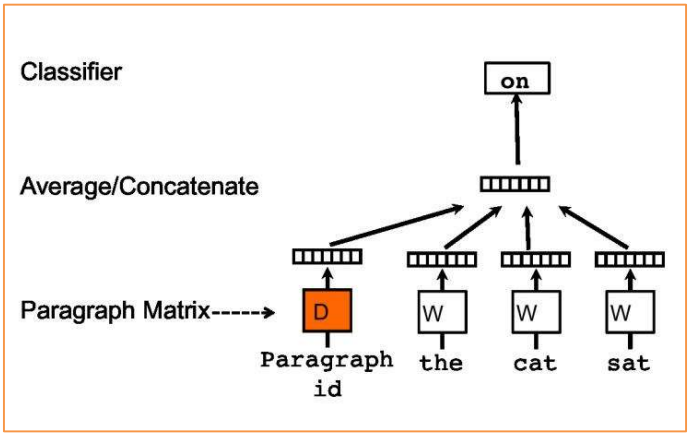


Ilustración 53: Estructura de Distributed Memory

De esta forma, al entrenar los vectores de cada palabra, también se entrena el vector del documento, obteniendo una representación vectorial del mismo una vez finalizado el entrenamiento.

Durante el entrenamiento, podemos pensar en el paragraph id como si fuese la representación vectorial de otra palabra, actúa como una memoria que recuerda lo que falta en el contexto actual.

El contexto es una ventana deslizante de tamaño fijo, comparte el paragraph id durante todo el documento.

7.2.5.2. Distributed Bag of Words

Distributed Bag of Words está basado en SkipGram. Esta variante no utiliza las palabras del contexto (ventana deslizante) en el input, sino que hace que el modelo prediga las palabras del documento en el output (Ilustración 54).

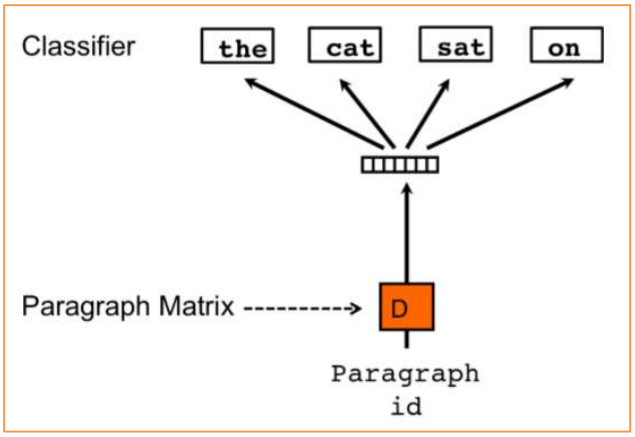


Ilustración 54: Estructura de Distributed Bag of Words

Además de ser conceptualmente más simple, esta variante tiene la ventaja de ser menos exigente en cuanto a almacenamiento, al no tener que almacenar los vectores de las palabras, solamente los pesos de la función Softmax.

7.3. Implementación

7.3.1. Modelos Basados en Bag of Words

7.3.1.1. Binary Counts

7.3.1.1.1. Creación del modelo

Comenzamos cargando el dataset en memoria como objeto Dataframe de la librería pandas. Como se indicó en el sprint 1, el preprocesado de los documentos es el mismo para todos los métodos a implementar y, al tratarse de una tarea costosa computacionalmente, se ha decidido exportar el dataset ya preprocesado para poder importarlo y comenzar a trabajar con él rápidamente.

```
[ ]  # Cargamos el dataset preprocesado
     df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/TFG/datasets/ticnn_preprocessed.csv')
```

Ilustración 55: Lectura del archivo csv

Mostramos las primeras cinco entradas del dataset para asegurarnos de que el dataset se ha cargado correctamente utilizando la función head():

```
[ ]  df.head()
```

	text	type
0	['donald', 'trump', 'property', 'showcase', 'b...	0.0
1	['trump', 'foundation', 'tell', 'new', 'york',...	0.0
2	['donald', 'trump', 'prepares', 'white', 'hous...	0.0
3	['lure', 'chinese', 'investor', 'trump', 'name...	0.0
4	['melania', 'barron', 'trump', 'wont', 'immedi...	0.0

Ilustración 56: Primeras entradas del dataframe

El dataset está ordenado de tal forma que la primera mitad son las noticias clasificadas como fiables y la segunda las fake, por lo que utilizamos la función sample de pandas para cambiar el orden aleatoriamente:

```
[ ]  df = df.sample(frac=1)
```

Ilustración 57: Cambiar el orden de las entradas

Guardamos los documentos y las clases a las que pertenecen como listas:

```
[ ]  texts = list(df['text'])
     targets = list(df['type'])
```

Ilustración 58: Guardar las entradas y las target variables como listas

Para entrenar el modelo, no se usará el conjunto de datos al completo, sino que se dividirá en una proporción definida, dedicando el 70% de los datos a entrenar el modelo (conjunto de entrenamiento) y el 30% restante se empleará como conjunto de pruebas para comprobar que el modelo funciona correctamente con datos desconocidos. Para realizar esta separación se ha definido la función train_test_split, que se muestra en la Ilustración 59:

```
[ ]  def train_test_split(texts, targets):
       # Dividimos en conjunto de entrenamiento y pruebas con una distribución de
       # un 70% de los datos para entrenamiento y un 30% para pruebas
       train_test_split = int(len(targets)*0.7)

       x_train = texts[:train_test_split]
       y_train = targets[:train_test_split]

       x_test = texts[train_test_split:]
       y_test = targets[train_test_split:]

       return x_train, y_train, x_test, y_test
```

Ilustración 59: Función train_test_split

Utilizamos dicha función para obtener los datos de entrenamiento y pruebas:

```
[ ]  x_train, y_train, x_test, y_test = train_test_split(texts, targets)
```

Ilustración 60: División en entrenamiento y pruebas

Creamos el objeto CountVectorizer llamando a la función correspondiente de Sklearn, se han empleado los siguientes parámetros [72]:

- **Binary:** De tipo boolean, se le asigna el valor True si se desea que almacene las apariciones binarias de cada palabra en lugar del número de apariciones.
- **Lowercase:** De tipo boolean, si su valor es true, convierte todas las palabras a minúscula. En este caso no es necesario porque los datos preprocesados ya han sido convertidos a minúscula.
- **Max_features:** De tipo entero. Determina el tamaño del vocabulario, quedándose con las n palabras más comunes del corpus. Este valor nos permite establecer un tamaño de entrada fijo para las redes neuronales.

```
[ ]  binary = CountVectorizer(binary = True, max_features = vocabulary_size, lowercase = False)
```

Ilustración 61: Creación de la matriz de incidencia binaria

Llamamos a la función fit_transform de Sklearn para que genere la matriz de incidencia del conjunto de entrenamiento. Esta función devuelve los vectores dispersos de cada uno de los documentos como un objeto de tipo sparse matrix del paquete scipy, que no puede usarse como entrada para los modelos neuronales.

Inicialmente, se convirtió a un objeto de tipo array de numpy, pero al tratarse de un conjunto formado por decenas de miles de vectores, cada uno de ellos de dimensión 20.000 (tamaño del vocabulario), se producía un Overflow de memoria, ya que la matriz de numpy ocupa alrededor de 10GB. Para solucionar este problema, se ha definido la función convert_to_sparse_tensor, que recibe como parámetro la matriz sparse de scipy y la convierte en un SparseTensor de la librería tensorflow, que sí puede emplearse como entrada de un modelo neuronal. Los SparseTensor optimizan considerablemente el espacio de almacenamiento al limitarse a guardar los índices de la matriz con un valor distinto de cero:

```
[ ]  # Obtenemos la representación densa (en forma de matrix)
     def convert_to_sparse_tensor(matrix):
       # Convertimos a numpy
       sparse_binary = matrix.todense()
       del matrix

       # Convertimos a tensor
       tensor_binary = tf.convert_to_tensor(sparse_binary, dtype = tf.float64)
       del sparse_binary

       # Convertimos a tensor sparse (menor almacenamiento)
       tensor_sparse = tf.sparse.from_dense(tensor_binary)
       del tensor_binary

       return tensor_sparse
```

Ilustración 62: Función convert_to_sparse_tensor

Aplicamos la función a la matriz sparse de scipy del conjunto de entrenamiento:

```
[ ]  # Obtenemos la representación densa del conjunto de entrenamiento
     tensor_sparse_binary_train = convert_to_sparse_tensor(sparse_binary_texts)
```

Ilustración 63: Conversión del conjunto de entrenamiento en SparseTensor

Para obtener la matriz de incidencia en el conjunto de pruebas utilizamos la función transform de Sklearn. Utilizamos transform y no fit_transform para que el vocabulario sea el mismo en ambos conjuntos. Por último, llamamos a la función convert_to_sparse_tensor para convertir la matriz de incidencia del conjunto de pruebas en un objeto de tipo SparseTensor.

```
[ ]  sparse_binary_texts = binary.transform(x_test)
     tensor_sparse_binary_test = convert_to_sparse_tensor(sparse_binary_texts)
```

Ilustración 64: Conversión del conjunto de pruebas en SparseTensor

7.3.1.1.2. Entrenamiento del Modelo

Una vez obtenidos los vectores dispersos de los documentos, utilizaremos la regresión logística para realizar la clasificación de los datos. Para crear el modelo, se define la función generate_logistic_regression():

```
[ ]  # Declaramos el modelo
     def generate_logistic_regression(dimension):
       # Modelo secuencial
       model = tf.keras.models.Sequential()

       # Capa de entrada
       model.add(tf.keras.layers.InputLayer(input_shape = (dimension, ),
                                            sparse = True,
                                            batch_size= 20,
                                            dtype = tf.float64))

       # Regresión logística
       model.add(tf.keras.layers.Dense(units=1, activation='sigmoid', dtype = tf.float64))

       # Compilamos el modelo
       model.compile(optimizer='rmsprop', loss='binary_crossentropy',
                     metrics=['accuracy', 'Precision', 'Recall', nlp_f.f1_score])

       return model
```

Ilustración 65: Función generate_logistic_regression

La capa de entrada es del tamaño del vocabulario (parámetro dimension) y recibe como parámetro un SparseTensor (debe indicarse con el parámetro sparse = True). La capa de regresión logística utiliza la función sigmoide como función de activación. Por último, se indican las métricas a emplear (accuracy, precision, recall y F1-score). Tensorflow no provee el cálculo de F1-score, por lo que lo calculamos a partir de los valores de precision y recall:

```
def f1_score(y_true, y_pred):
    true_positives = k.sum(k.round(k.clip(y_true * y_pred, 0, 1)))
    possible_positives = k.sum(k.round(k.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + k.epsilon())

    predicted_positives = k.sum(k.round(k.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + k.epsilon())

    return 2 * ((precision * recall) / (precision + recall + k.epsilon()))
```

Ilustración 66: Función f1_score

El modelo cuenta con un total de 20.001 variables a optimizar. Es un número muy elevado teniendo en cuenta la relativa simplicidad del modelo, especialmente sabiendo que este aumenta considerablemente al añadir más capas.

```
[ ]  model.summary()

     Model: "sequential"
     _________________________________________________________________
     Layer (type)                 Output Shape              Param #
     =================================================================
     dense (Dense)                (20, 1)                   20001
     =================================================================
     Total params: 20,001
     Trainable params: 20,001
     Non-trainable params: 0
     _________________________________________________________________
```

Ilustración 67:Variables del modelo

Por último, entrenamos el modelo durante 10 épocas:

```
history = model.fit(tensor_sparse_binary_train,
                    tf.convert_to_tensor(y_train, dtype = tf.float64),
                    epochs = 10,
                    validation_data = (tensor_sparse_binary_test,
                                       tf.convert_to_tensor(y_test,
                                                            dtype = tf.float64)),
                    callbacks = [tensorboard_callback])
```

Ilustración 68: Entrenamiento del modelo

7.3.1.1.3. Evaluación del modelo

Para la evaluación del modelo, se utilizarán gráficas que muestren las variaciones de cada una de las métricas a lo largo de las épocas en las que se entrena el modelo, tanto en el conjunto de entrenamiento como en el de pruebas. Para obtener dichas representaciones se ha definido la función plot_history:

```python
def plot_history(history, accuracy=True, precision=True, recall=True,
                 f1=True, loss=True):
    plt.style.use('ggplot')

    # Accuracy
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']

    x = range(1, len(acc) + 1)

    if accuracy:
        plt.ylim(0., 1.)
        plt.plot(x, acc, 'b', label='Accuracy en entrenamiento')
        plt.plot(x, val_acc, 'r', label='Accuracy en pruebas')
        plt.title('Accuracy en entrenamiento y pruebas')
        plt.legend()

        plt.show()
```

Ilustración 69: Función plot_history

También es necesario comprobar si la estructura del modelo neuronal es la esperada, para ello utilizamos la librería tensorboard para obtener una representación gráfica del grafo de operaciones, en el que podemos observar como la entrada (input_1) pasa por la capa densa (en este caso se trata de un solo perceptrón) y envía los resultados a metrics y loss, donde se calculan las métricas (accuracy, precision, recall y F1-score) y la función de pérdidas respectivamente:

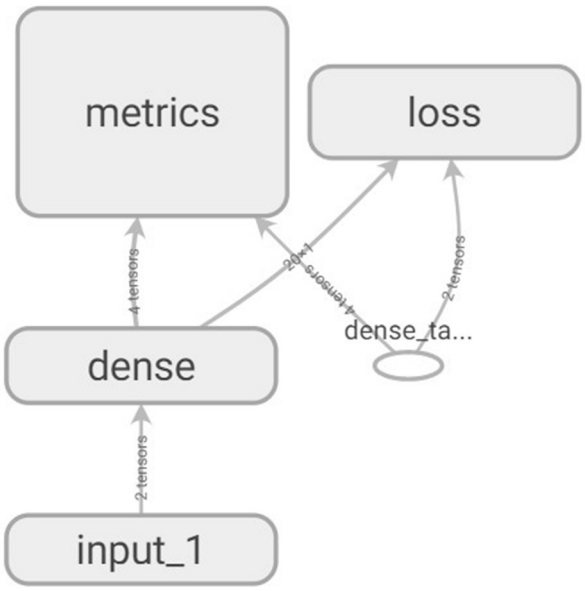


Ilustración 70: Grafo de operaciones Binary Count

7.3.1.2. Term Frequency

La implementación de Term Frequency es idéntica a la de Binary Counts a excepción del vector utilizado para generar la matriz de incidencia. En este caso aplicamos TfIdfVectorizer de la librería Sklearn, en lugar de utilizar CountVectorizer [73].

```
[ ]  tf_vector = TfidfVectorizer(max_features = vocabulary_size, lowercase = False, use_idf = False)
```

Ilustración 71: Creación del vector Term Frequency

En la Ilustración 71 se muestra la llamada a la función TfIdfVectorizer, todos los parámetros son idénticos a los utilizados en Binary Counts, exceptuando use_idf, de tipo boolean. Si este parámetro es False, se utiliza Term Frequency, en caso contrario, se utiliza tf-idf.

7.3.1.3. Term Frequency – Inverse Document Frequency

De nuevo, la implementación de tf-idf es idéntica a la de Binary Counts excepto por el vector utilizado. De nuevo, se llama a la función TfidfVectorizer, pero esta vez se indica que el parámetro use_idf es True [73].

```
[ ]  tfidf = TfidfVectorizer(lowercase = False, max_features = vocabulary_size, use_idf = True)
```

Ilustración 72: Creación del vector tf-idf

7.3.2. Naive Bayes

7.3.2.1. Creación del modelo

Cargamos el dataset preprocesado como objeto Dataframe de la librería pandas:

```
[ ]  %%time
     df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/TFG/datasets/ticnn_preprocessed.csv')
```

Ilustración 73: Lectura del archivo csv

Mostramos las primeras cinco entradas del dataset para asegurarnos de que el dataset se ha cargado correctamente utilizando la función head:

```
[ ]  df.head()
```

	text	type
0	['donald', 'trump', 'property', 'showcase', 'b...	0.0
1	['trump', 'foundation', 'tell', 'new', 'york',...	0.0
2	['donald', 'trump', 'prepares', 'white', 'hous...	0.0
3	['lure', 'chinese', 'investor', 'trump', 'name...	0.0
4	['melania', 'barron', 'trump', 'wont', 'immedi...	0.0

Ilustración 74: Primeras cinco entradas del Dataframe

Utilizamos la función sample de pandas para cambiar el orden aleatoriamente

```
[ ]  df = df.sample(frac=1)
```

Ilustración 75: Cambio del orden de las entradas

Guardamos los documentos y las clases a las que pertenecen como listas:

```
[ ]  texts = list(df['text'])
     targets = list(df['type'])
```

Ilustración 76: Guardar las entradas y las target variables como listas

Dividimos en conjunto de entrenamiento y pruebas utilizando la función train_test_split explicada en la implementación de Binary Counts. Destinamos un 70% del dataset a entrenamiento y un 30% a pruebas.

```
[ ]  x_train, y_train, x_test, y_test = train_test_split(texts, targets)
```

Ilustración 77: División en conjunto de entrenamiento y pruebas

Para poder estimar las probabilidades condicionales en Naive Bayes, es necesario pasarle un CountVectorizer como parámetro:

```
[ ]  count = CountVectorizer(lowercase = False, binary = False)
```

Ilustración 78: Creación del vector CountVectorizer

No establecemos un parámetro max_features porque en Naive Bayes no hay un tamaño de input fijo. Establecemos el valor del parámetro binary como false para que cuente el número total de apariciones de un término en un documento dado. Llamamos a la función fit_transform para

obtener la matriz de incidencia en el conjunto de entrenamiento, y a la función transform para obtener la del conjunto de pruebas:

```
[ ]  count_x_train = count.fit_transform(x_train)
     count_x_test = count.transform(x_test)
```

Ilustración 79: Funciones fit_transform y transform

7.3.2.2. Entrenamiento del modelo

Llamamos a la función MultinomialNB del paquete naive_bayes de Sklearn e invocamos a la función fit, que estima las probabilidades en el conjunto de entrenamiento.

```
[ ]  %%time
     naive_bayes = MultinomialNB()
     naive_bayes.fit(count_x_train, y_train)
```

Ilustración 80: Creación del objeto de tipo MultinomialNB

7.3.2.3. Evaluación del modelo

Al tratarse de un modelo que no se entrena por épocas, no podemos utilizar la función plot_history para evaluar el rendimiento, así que se utilizará una matriz de confusión.

Comenzamos llamando a la función predict, pasándole el conjunto de pruebas como parámetro.

```
[ ]  predictions = naive_bayes.predict(count_x_test)
```

Ilustración 81: Predicción del modelo para el conjunto de pruebas

Mostramos el valor de cada una de las métricas importándolas del paquete metrics de Sklearn (reciben los valores reales y las predicciones como parámetros).

```
[ ]  print('Accuracy: ', accuracy_score(y_test, predictions))
     print('Precision: ', precision_score(y_test, predictions))
     print('Recall: ', recall_score(y_test, predictions))
     print('F1-Score: ', f1_score(y_test, predictions))

     Accuracy:  0.8708333333333333
     Precision:  0.8616852146263911
     Recall:  0.8885245901639345
     F1-Score:  0.8748991121872478
```

Ilustración 82: Métricas del modelo en el conjunto de pruebas

Por último, representamos los valores de forma más visual con una matriz de confusión utilizando la librería seaborn:

```
[ ]  cm = confusion_matrix(y_test, predictions)
     sns.heatmap(cm, square = True, annot = True,
                 cmap = 'RdBu', cbar = False,
                 xticklabels = ['Reliable', 'Fake'],
                 yticklabels = ['Reliable', 'Fake'])

     plt.xlabel('true label')
     plt.ylabel('predicted label')
     plt.show()
```

Ilustración 83: Creación de matriz de confusión usando seaborn

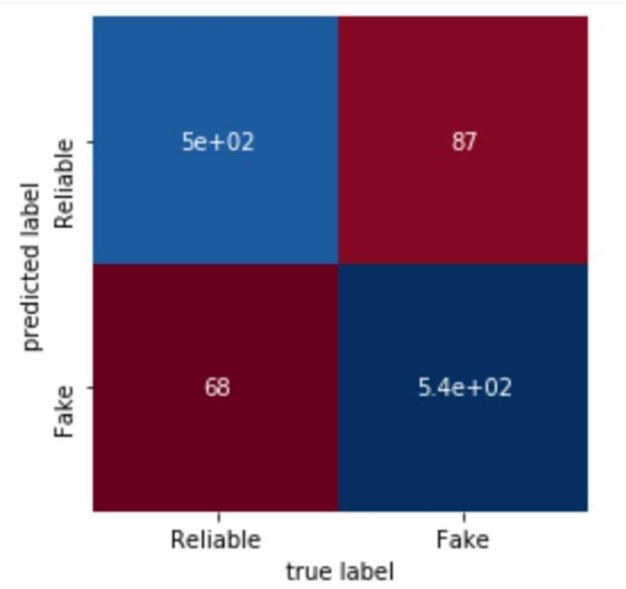


Ilustración 84: Ejemplo de matriz de confusión generada utilizando seaborn

7.3.3. Word2Vec (SkipGram y Continuous Bag of Words)

7.3.3.1. Creación del modelo

Comenzamos cargando el dataset preprocesado como objeto Dataframe de la librería pandas

```
[ ]  # Cargamos el dataset preprocesado
     df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/TFG/datasets/ticnn_preprocessed.csv')
```

Ilustración 85: Lectura del fichero csv

Mostramos las cinco primeras entradas del dataset para asegurarnos de que el conjunto de datos se ha cargado correctamente utilizando la función head:

```
[ ]  df.head()
```

	text	type
0	['donald', 'trump', 'property', 'showcase', 'b...	0.0
1	['trump', 'foundation', 'tell', 'new', 'york',...	0.0
2	['donald', 'trump', 'prepares', 'white', 'hous...	0.0
3	['lure', 'chinese', 'investor', 'trump', 'name...	0.0
4	['melania', 'barron', 'trump', 'wont', 'immedi...	0.0

Ilustración 86: Cinco primeras entradas del dataframe

La columna text está almacenada como string a pesar de tratarse de un objeto de tipo lista, las funciones encargadas de procesar los modelos basados en Bag of Words y Naive Bayes se encargaban automáticamente de gestionar estos casos, pero para los modelos de Word2Vec es necesario volver a convertir el contenido en un objeto de tipo string, utilizando para ello literal_eval de la librería ast:

```
[ ]  from ast import literal_eval
     df['text'] = df['text'].apply(literal_eval)
```

Ilustración 87: Conversión de string a lista

Utilizamos la función sample de pandas para cambiar el orden aleatoriamente:

```
[ ]  df = df.sample(frac=1)
```

Ilustración 88: Cambio del orden de las entradas

Guardamos los documentos y las variables target como listas:

```
[ ]  texts = list(df['text'])
     targets = list(df['type'])
```

Ilustración 89: Guardar las entradas y las target variables como listas

Dividimos en conjunto de entrenamiento y pruebas llamando a la función train_test_split (70% del conjunto para entrenamiento y 30% restante para pruebas):

```
[ ]  x_train, y_train, x_test, y_test = train_test_split(texts, targets)
```

Ilustración 90: Separación en conjunto de entrenamiento y pruebas

7.3.3.2. Entrenamiento del modelo

Declaramos el modelo Word2Vec llamando a la función Word2Vec de la librería gensim indicando los siguientes parámetros [74]:

- **Size:** De tipo entero, es el número de dimensiones del Word embedding (el valor por defecto es 100, pero el recomendado es 300)
- **Window:** De tipo entero, distancia máxima entre la palabra central y las que la rodean (contexto). El valor por defecto es 5.
- **Min_count:** De tipo entero. Número mínimo de apariciones de cada palabra, si una palabra tiene un total de apariciones en todo el corpus inferior a este valor, será ignorada.
- **Workers:** De tipo entero. Número de particiones durante el entrenamiento. Valor por defecto: 3.
- **Sg:** Algoritmo de entrenamiento. Si vale 0 se aplica Continuous Bag of Words, si vale 1 se aplica SkipGram.
- **Iter:** Número de épocas.

```
[ ]  %%time
     model = Word2Vec(x_train, min_count = 5, size = 300,
                      window = 5,
                      iter = 10,
                      sg = 1)
```

Ilustración 91: Creación del modelo Word2Vec

Expresamos cada documento como la media de todos sus Word-embeddings, para ello definimos la función Word_average:

```
[ ]  def word_average(doc, model):
         mean = []
         for word in doc:
           if word in model.wv.vocab:
             mean.append(model.wv.get_vector(word))

         if not mean:
           # Si el texto está vacío, devuelve un vector de ceros
           return np.zeros(model.vector_size)
         else:
           mean = np.array(mean).mean(axis=0)
           return mean
```

Ilustración 92: Función word_average

Aplicamos esta función a todos los documentos del conjunto de entrenamiento y pruebas:

```
[ ]  %%time
     x_train_dense = []
     for doc in x_train:
       dense_doc = word_average(doc, model)
       x_train_dense.append(dense_doc)
```

Ilustración 93: Aplicación de word_average al conjunto de entrenamiento

Una vez obtenidas las representaciones vectoriales de cada uno de los documentos, utilizamos una red neuronal simple consistente en:

- Una capa fully connected con función de activación relu
- Una capa de Dropout con rate 0.2
- Una capa fully connected de una sola neurona y función de activación sigmoide.

Para crear los modelos, definimos la función generate_w2v_dense_nn:

```
[ ]  # Declaramos el modelo
     def generate_w2v_dense_nn():
       # Modelo secuencial
       model = tf.keras.models.Sequential()

       # Capa de entrada
       model.add(tf.keras.layers.InputLayer(input_shape = (300, ),
                                            dtype = tf.float32))

       # Capa densa
       model.add(
           tf.keras.layers.Dense(
               units = 150,
               activation = 'relu'
           )
       )

       # Capa Dropout
       model.add(
           tf.keras.layers.Dropout(
               rate = 0.2
           )
       )

       # Regresión logística
       model.add(tf.keras.layers.Dense(units=1, activation='sigmoid', dtype = tf.float32))

       # Compilamos el modelo
       model.compile(optimizer='rmsprop', loss='binary_crossentropy',
                     metrics=['accuracy', 'Precision', 'Recall', nlp_f.f1_score])
```

Ilustración 94: Función generate_w2v_logistic_regression

La capa de entrada es de tamaño 300 (número de dimensiones de los Word_embeddings). El modelo cuenta con 45.301 variables a optimizar, este número de variables es superior al de los modelos basados en Bag of Words, pero esto se debe a que tiene un mayor número de capas intermedias. En el caso de que se utilizase simplemente la regresión logística, como es el caso de dichos modelos, el número de variables sería de 301. Este número de variables es bastante inferior al de los modelos basados en Bag of Words, a pesar de que estos vectores contienen mucha más información referente al contexto de cada palabra.

```
[ ] model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 150)               45150
_________________________________________________________________
dropout (Dropout)            (None, 150)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 151
=================================================================
Total params: 45,301
Trainable params: 45,301
Non-trainable params: 0
_________________________________________________________________
```

Ilustración 95: Variables de los modelos Word2Vec

Por último, entrenamos el modelo durante 10 épocas:

```
history = model.fit(np.array(x_train_dense),
                    np.array(y_train),
                    epochs = 10,
                    validation_data = (np.array(x_test_dense), np.array(y_test)),
                    callbacks = [tensorboard_callback])
```

Ilustración 96: Entrenamiento del modelo Word2Vec

7.3.3.3. Evaluación del modelo

Para la evaluación del modelo, volvemos a utilizar la función de plot_history, explicada en la implementación de Binary Counts:

Por último, obtenemos el grafo de computación utilizando TensorBoard. Gracias a este grafo podemos observar que esta vez el modelo está compuesto por dos capas densas (dense y dense_1), y que a la primera capa densa se le aplica dropout.

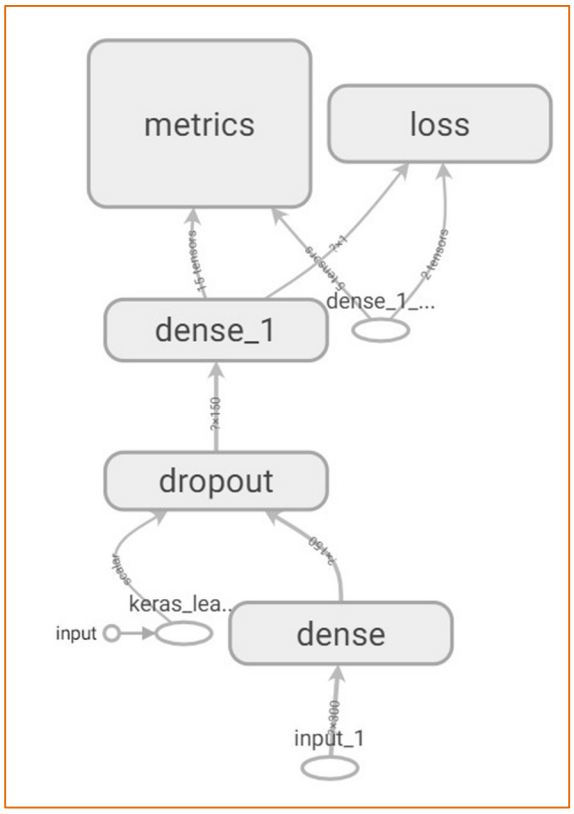


Ilustración 97: Grafo de Computación para Word2Vec

7.3.4. Doc2Vec (Distributed Memory y Distributed Bag of Words)

7.3.4.1. Creación del modelo

De nuevo, comenzamos cargando el conjunto de datos en memoria como objeto Dataframe de pandas. Al igual que ocurría en la implementación de Word2Vec, la columna "text" del dataset es leída como un string en lugar de como una lista de strings, por lo que aplicamos literal_eval para cambiar el formato de esta columna.

A continuación, alteramos aleatoriamente el orden de las entradas del dataset y dividimos en entrenamiento y pruebas utilizando la función train_test_split. Todos estos pasos se muestran en la Ilustración 98:

```
[ ]  # Lectura del dataset
     df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/TFG/datasets/fakeAndReliable_balanced.csv')

     # Convertir la columna "content" a lista de Strings
     from ast import literal_eval
     df['content'] = df['content'].apply(literal_eval)

     # Reordenar las entradas aleatoriamente
     df = df.sample(frac=1)

     # Convertir los documentos y las target variables a listas
     texts = list(df['content'])
     targets = list(df['type'])

     # División en conjunto de entrenamiento y pruebas
     x_train, y_train, x_test, y_test = train_test_split(texts, targets)
```

Ilustración 98: Lectura del Dataset

La función que inicializa el modelo Doc2Vec recibe como parámetro una lista de objetos de tipo TaggedDocument, estos objetos consisten en una tupla formada por dos elementos:

- Lista ordenada de los términos que pertenecen al modelo
- Tag del documento (1.0 si es una fake new y 0.0 si es una noticia fiable)

La creación de objetos de tipo TaggedDocument para los conjuntos de entrenamiento y pruebas se muestra en la Ilustración 99:

```
[14] train_documents = []
     test_documents = []

     for i in range(len(x_train)):
       train_documents.append(
           TaggedDocument(words = x_train[i], tags = [y_train[i]])
           )

     for i in range(len(x_test)):
       test_documents.append(
           TaggedDocument(words = x_test[i], tags = [y_test[i]])
       )
```

Ilustración 99: Creación de los objetos de tipo TaggedDocument

7.3.4.2. Entrenamiento del modelo

Declaramos el modelo Doc2Vec llamando a la función Doc2Vec de la librería gensim indicando los siguientes parámetros [75]:

- **Dm:** Define el algoritmo de entrenamiento. Si vale 1, se aplica Distributed Memory, mientras que, si vale 0, se aplica Distributed Bag of Words.
- **Vector_size:** Número de dimensiones de los vectores que representan a los documentos.
- **Min_count:** Ignora todas las palabras con una frecuencia total inferior al valor de este parámetro.

- **Workers:** Número de hilos utilizados para entrenar el modelo.

```
[ ]  distributed_memory = Doc2Vec(dm = 1,
                                  vector_size = 300,
                                  min_count = 2,
                                  workers = cores
                                  )
```

Ilustración 100: Creación del Modelo Doc2Vec

Definimos el vocabulario del modelo llamando a la función build_vocab y pasándole por parámetro el conjunto de entrenamiento:

```
[ ]  distributed_memory.build_vocab([x for x in train_documents])
```

Ilustración 101: Definición del Vocabulario del modelo

Entrenamos el modelo durante 10 épocas:

```
[21] %%time
     distributed_memory.train(train_documents,
                              total_examples = len(train_documents),
                              epochs = 10)
```

Ilustración 102: Entrenamiento del modelo Doc2Vec

Para entrenar el modelo neuronal debemos separar el vector de cada documento de su clasificación, además de generar las representaciones vectoriales de cada uno de ellos en base al modelo de Doc2Vec entrenado previamente, para ello definimos la función vector_for_learning (Ilustración 103), que recibe como parámetros una lista de TaggedDocuments y un modelo de Doc2Vec entrenado y devuelve una lista con las representaciones vectoriales de cada uno de los documentos recibidos por parámetro, además de la lista de target variables.

```
[ ]  def vector_for_learning(model, docs):
       sents = docs
       targets, feature_vectors = zip(*[(doc.tags[0],
                                         model.infer_vector(doc.words, steps=20)) for doc in sents])

       return targets, feature_vectors
```

Ilustración 103: Función vector_for_learning

Llamamos a esta función para obtener los conjuntos de entrenamiento y pruebas en el formato necesario para el entrenamiento de la red neuronal.

```
[ ]  y_train, x_train = vector_for_learning(distributed_memory, train_documents)

     y_test, x_test = vector_for_learning(distributed_memory, test_documents)
```

Ilustración 104: Aplicación de la función vector_for_learning a los conjuntos de entrenamiento y pruebas

Al igual que en Word2Vec, empleamos una red neuronal con tres capas (dos densas y una de dropout) para realizar la clasificación:

```
[29] model.summary()

     Model: "sequential"
     _________________________________________________________________
     Layer (type)                 Output Shape              Param #
     =================================================================
     dense (Dense)                (None, 150)               45150
     _________________________________________________________________
     dropout (Dropout)            (None, 150)               0
     _________________________________________________________________
     dense_1 (Dense)              (None, 1)                 151
     =================================================================
     Total params: 45,301
     Trainable params: 45,301
     Non-trainable params: 0
     _________________________________________________________________
```

Ilustración 105: Variables del modelo neuronal

Entrenamos el modelo durante 10 épocas

```
history = model.fit(np.array(x_train),
                    np.array(y_train),
                    epochs = 10,
                    validation_data = (np.array(x_test), np.array(y_test)),
                    callbacks = [tensorboard_callback])
```

Ilustración 106: Entrenamiento del modelo neuronal

7.3.4.3. Evaluación del modelo

Para la evaluación del modelo, volvemos a utilizar la función plot_history. La representación del grafo de computación, obtenida utilizando TensorBoard, es la siguiente:

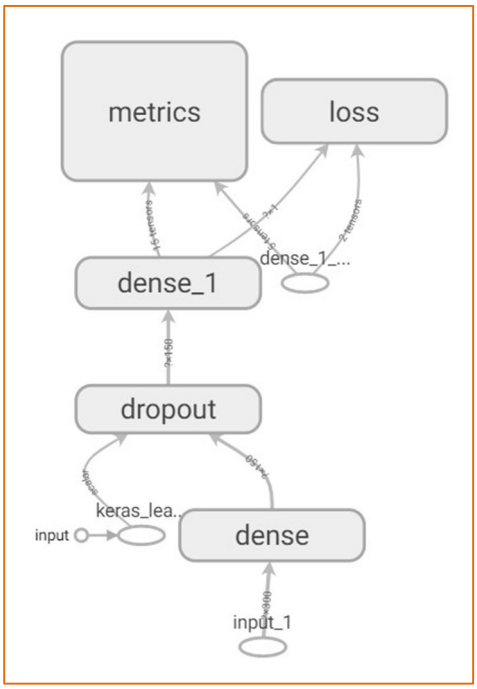


Ilustración 107: Grafo de Computación Doc2Vec

7.4. Análisis de los resultados

7.4.1. Bag of Words

Los modelos basados en Bag of Words son los métodos de clasificación más simples dentro del campo de procesamiento del lenguaje natural, ya que solo tienen en cuenta si las palabras del vocabulario aparecen o no en un texto para llevar a cabo la clasificación, sin considerar el orden de aparición de las palabras o el contexto de estas. Por estos motivos, suelen esperarse resultados mediocres (En torno al 60% de precisión en el caso de un dataset balanceado), especialmente si tenemos en cuenta que estos artículos ya han sido utilizados en publicaciones anteriores.

No obstante, al entrenar el modelo con un vocabulario de 20.000 palabras, podemos observar que el F1-Score en el conjunto de pruebas alcanza el 95% en Binary Counts (Tabla 42), y el 93% en term frequency (Tabla 43) y en tf-idf (Tabla 44). Para experimentar con estos resultados, se ha reducido progresivamente el tamaño del vocabulario para comprobar si los modelos siguen convergiendo de esta forma.

Antes de continuar analizando los resultados, es importante recordar que la distribución de las clases no está completamente equilibrada, ya que el 58% de las noticias son falsas y el 42% restante son reales, esta proporción es adecuada para realizar el entrenamiento, pero a la hora de interpretar las métricas hay que tener en cuenta que el peor clasificador posible (clase mayoritaria), tendrá una accuracy del 58%, no del 50%.

	BOW-Binary Counts					
	# Palabras	F1-Score	Recall	Precision	Accuracy	Loss
Entrenamiento	20.000	0.989	0.9947	0.9843	0.9843	0.0236
Pruebas		0.9592	0.9782	0.9445	0.9538	34.95
Entrenamiento	1.000	0.9576	0.9735	0.9454	0.9515	0.1815
Pruebas		0.9367	0.9522	0.9281	0.9293	23.79
Entrenamiento	100	0.827	0.8557	0.8118	0.8008	0.3874
Pruebas		0.8176	0.8374	0.8121	0.7894	16.302
Entrenamiento	25	0.7763	0.7979	0.7701	0.7407	0.544
Pruebas		0.7666	0.7890	0.7657	0.7405	6.6268

Tabla 42: Resultados de Binary Counts en función del tamaño del vocabulario

En el caso de Binary Counts (Tabla 42), el valor de F1-Score sigue superando el 90% con un vocabulario de 1000 palabras. Al reducirlo a 100 y a 25 palabras, el valor de esta métrica se reduce a 81% y a 76% respectivamente. En estos casos, es posible imprimir el vocabulario y buscar patrones. Observando dicho vocabulario, podemos descubrir que está compuesto principalmente por palabras referentes a lo que parecen ser las elecciones a la presidencia de los Estados Unidos del año 2016 (american, campaign, Clinton, election, Hillary, president, republican, state, Trump, vote), además de otras palabras comunes (also, go, get, take, entre otras).

Como conclusión, podemos deducir que los datos poseen un sesgo importante referente a las noticias relativas a las elecciones, lo que explica en cierta medida los buenos resultados

obtenidos por los artículos originales. Las noticias reales, a pesar de pertenecer al mismo período, tratan un espectro más amplio de noticias.

	BOW-Term Frequency					
	# Palabras	F1-Score	Recall	Precision	Accuracy	Loss
Entrenamiento	20.000	0.9405	0.959	0.9266	0.9316	0.1694
Pruebas		0.9304	0.9518	0.9154	0.9205	50.9067
Entrenamiento	1.000	0.92	0.9379	0.9089	0.9086	0.2407
Pruebas		0.9119	0.9325	0.8978	0.8989	35.3047
Entrenamiento	100	0.8572	0.8813	0.8437	0.8351	0.406
Pruebas		0.8483	0.8708	0.8375	0.8262	13.8539
Entrenamiento	25	0.8252	0.8521	0.8116	0.798	0.5003
Pruebas		0.824	0.8529	0.8124	0.7989	5.2959

Tabla 43:Resultados de tf en función del tamaño del vocabulario

En lo referente a Term Frequency (Tabla 43), los resultados son un poco peores que en Binary Counts (lo cual resulta extraño al tratarse de un modelo ligeramente superior). También podemos notar que el desequilibrio entre recall y precision es mayor, lo que denota que se decanta por predecir la clase mayoritaria.

	BOW-Term Frequency-Inverse Document Frequency					
	# Palabras	F1-Score	Recall	Precision	Accuracy	Loss
Entrenamiento	20.000	0.9404	0.9633	0.9248	0.9326	0.2504
Pruebas		0.9326	0.953	0.9165	0.9223	52.0268
Entrenamiento	1.000	0.9211	0.944	0.9056	0.9095	0.1943
Pruebas		0.9109	0.9325	0.8969	0.8987	35.2181
Entrenamiento	100	0.8559	0.8823	0.8404	0.8342	0.3848
Pruebas		0.8572	0.886	0.8397	0.8325	13.4264
Entrenamiento	25	0.8277	0.8642	0.8054	0.7975	0.4589
Pruebas		0.8089	0.8934	0.7504	0.7673	5.3099

Tabla 44: Resultados de tf-idf en función del tamaño del vocabulario

Algo parecido ocurre con tf-idf (Tabla 44), que muestra una diferencia aún más notoria entre precision y recall, especialmente al reducir el tamaño del vocabulario, siendo el caso más extremo el del vocabulario de 25 palabras, con una diferencia del 14% entre ambas métricas.

Uno de los principales defectos de Binary Counts con respecto a las versiones más avanzadas de Bag of Words es que concede la misma importancia a todas las palabras (1 ó 0 si aparecen o no respectivamente), por lo que se caracteriza por ser muy sensible a las diferencias de longitud entre textos. Un desequilibrio en este aspecto podría explicar que los resultados sean ligeramente superiores.

La media aritmética de la longitud de cada tipo de noticia parece indicar que la diferencia de longitud no es muy notoria. Sin embargo, la media aritmética en ocasiones puede ser engañosa si la distribución de los valores está muy dispersa. Por este motivo, se han calculado los percentiles y la mediana (percentil 0,5), que demuestran que la longitud de ambos tipos de noticia no es tan similar como indica la media.

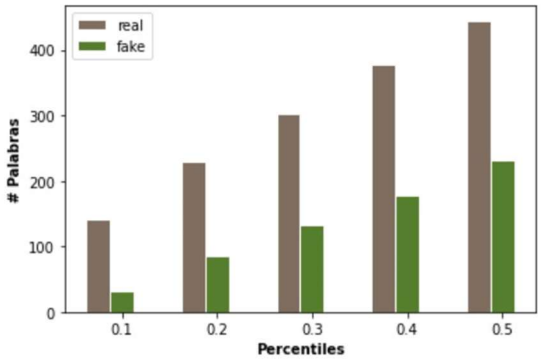


Ilustración 108: Percentiles 0.1 a 0.5

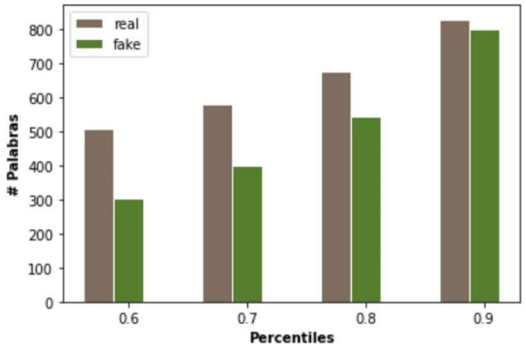


Ilustración 109: Percentiles 0.6 a 0.9

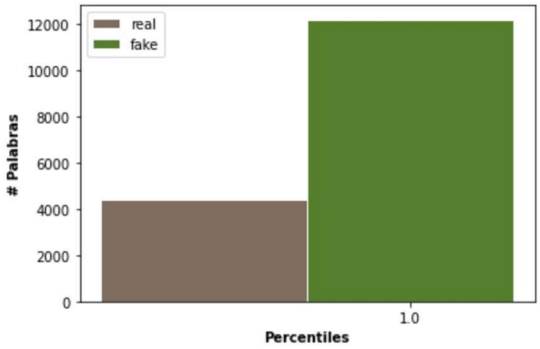


Ilustración 110: Percentil 1.0

La primera gráfica (Ilustración 108) muestra que en el primer percentil (0,1) las noticias reales son 4,6 veces más largas que las falsas. En los percentiles del 0,2 al 0,5 la diferencia es de 2 (casi tres en el percentil 0,2). Las noticias reales siguen siendo más largas hasta el percentil 0,8, equilibrándose en el 0,9.

El percentil 1 es el que hace que la media aritmética sea más o menos equilibrada, ya que aparecen una pequeña cantidad de noticias falsas de gran longitud (pero que no representan ni el 5% del total de los datos) que ajustan la media. Las proporciones entre los dos percentiles son las mostradas en la Ilustración 111.

Por los motivos expuestos en párrafos anteriores, podemos concluir que los clasificadores se ven afectados por dos sesgos muy importantes (longitud de las noticias y vocabulario perteneciente a las elecciones), que afectan especialmente a este tipo de métodos de clasificación. Estos métodos se apoyan (en mayor o menor medida) en la longitud de los textos para crear un hiperplano separador entre vectores con mayor o menor cantidad de dispersión.

	length real	length fake	proportion
0.1	142.0	31.0	4.580645
0.2	230.0	85.0	2.705882
0.3	302.0	132.0	2.287879
0.4	378.0	177.0	2.135593
0.5	445.0	232.0	1.918103
0.6	507.0	303.0	1.673267
0.7	579.0	401.0	1.443890
0.8	675.4	542.4	1.245206
0.9	828.0	797.0	1.038896
1.0	4439.0	12212.0	0.363495

Ilustración 111: Diferencias de proporción entre percentiles

Esto se ve reflejado en las funciones de pérdidas de entrenamiento y pruebas, como la de la Ilustración 112, que muestra órdenes de magnitud completamente diferentes, teniendo un valor de 0,25 en entrenamiento y de 35 en pruebas.

Ilustración 112: Vectores de pérdidas en Binary Counts

7.4.2. Naive Bayes

Naive Bayes es un modelo probabilístico que, dado un texto formado por un conjunto de palabras, determina la probabilidad de que dicho texto pertenezca a una categoría o a otra en función de la probabilidad individual de cada una de las palabras de pertenecer a cada categoría. El principal defecto de este método es que asume independencia entre los términos que componen un texto.

Considerando el vocabulario completo, el F1-Score de Naive Bayes es del 89%, no es un valor que resulte demasiado sospechoso, ya que el rendimiento de este baseline suele ser muy bueno

en la práctica a pesar de su aparente simplicidad. No obstante, para asegurarnos, procedemos a reducir el vocabulario para observar el comportamiento del modelo.

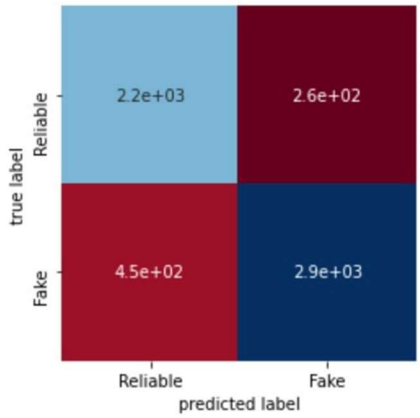


Tabla 45: Matriz de Confusión Naive Bayes (Vocabulario completo)

Al reducir el vocabulario a las 20.000 palabras más comunes, los resultados siguen siendo similares. No obstante, al reducirlo a 1.000, 100 y 5 palabras los resultados, a pesar de seguir siendo buenos, se decantan por la clase mayoritaria para realizar la clasificación. Con el vocabulario reducido, los términos son pertenecientes a las elecciones.

Naive Bayes				
# Palabras	**F1-Score**	**Recall**	**Precision**	**Accuracy**
Vocabulario completo	0.8916	0.8664	0.9184	0.8779
20.000	0.8896	0.8635	0.9173	0.8733
1.000	0.8613	0.8536	0.8692	0.8398
100	0.8292	0.8323	0.8262	0.801
10	0.7838	0.8273	0.7447	0.7319
5	0.725	0.7428	0.7081	0.6786

Tabla 46: Resultados de Naive Bayes en función del tamaño del vocabulario

Este modelo se ve menos afectado por el sesgo de la longitud de las noticias, al tratarse de un cálculo probabilístico, pero sigue viéndose afectado por el vocabulario de las elecciones, al no considerar elementos como el contexto, el orden o las dependencias entre términos.

7.4.3. Word2Vec

Los dos modelos basados en Word2Vec devuelven resultados casi idénticos, con un F1-Score del 92% y con valores equilibrados de precision y recall, algo que no ocurría en los modelos anteriores.

	Word2Vec				
	SkipGram				
	F1-Score	**Recall**	**Precision**	**Accuracy**	**Loss**
Entrenamiento	0.9265	0.9303	0.929	0.9174	0.2044
Pruebas	0.9215	0.9319	0.917	0.9116	0.2203
	Continuous Bag of Words				
	F1-Score	**Recall**	**Precision**	**Accuracy**	**Loss**
Entrenamiento	0.94	0.9418	0.9426	0.9327	0.1741
Pruebas	0.922	0.9385	0.9124	0.9108	0.2305

Tabla 47: Resultados de Word2Vec

No obstante, a pesar de que ambos modelos tienen un rendimiento casi idéntico, las Ilustraciones 113 y 114 demuestran que la función de pérdidas en el caso de SkipGram se comporta mejor que en Continuous Bag of Words ya que, en este último, la función de pérdidas en el conjunto de pruebas diverge considerablemente a partir de la época 8, lo que es un signo claro de Overfitting. En ambos casos se ha escogido 8 como el número de épocas de entrenamiento, los resultados de dicha época son los reflejados en la Tabla 47.

Ilustración 113: Función de pérdidas SkipGram

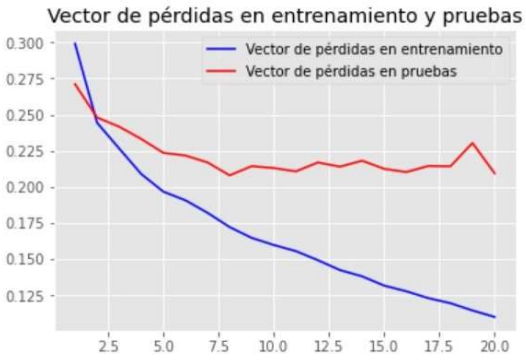


Ilustración 114: Función de pérdidas CBOW

En el caso de estos modelos, cada una de las entradas es la media de los Word Embeddings de las palabras que componen la noticia en cuestión, por lo que la influencia del sesgo provocado por la diferencia de longitud entre las noticias es bastante menor en este modelo en comparación con Naive Bayes y los modelos basados en Bag of Words, aunque sigue produciéndose en cierta medida, ya que realizar la media aritmética entre más elementos sigue pudiendo producir una diferenciación por longitud, aunque bastante menor que en los casos anteriores.

7.4.4. Doc2Vec

Las dos versiones de Doc2Vec devuelven resultados muy buenos, con un F1-Score del 92% en Distributed Memory y de un 94% en el caso de Distributed Bag of Words, los resultados de DBOW son ligeramente superiores, además de contar con un mejor equilibrio entre precision y recall, por lo que podemos considerarlo el mejor modelo en este caso.

	Doc2Vec				
	Distributed Memory				
	F1-Score	**Recall**	**Precision**	**Accuracy**	**Loss**
Entrenamiento	0.9641	0.9775	0.9534	0.9592	0.1007
Pruebas	0.9262	0.9531	0.9049	0.9128	0.2918
	Distributed Bag of Words				
	F1-Score	**Recall**	**Precision**	**Accuracy**	**Loss**
Entrenamiento	0.9764	0.9808	0.9727	0.9726	0.0736
Pruebas	0.9435	0.9405	0.9511	0.9372	0.1726

Tabla 48:Resultados de Doc2Vec

Como se muestra en las Ilustraciones 115 y 116, ambos modelos convergen muy rápidamente y divergen a partir de la época 4 en el conjunto de pruebas, por lo que serán entrenados durante este número de épocas. Los resultados de la Tabla 48 son de modelos entrenados durante 4 épocas.

Ilustración 115: Función de pérdidas DM

Ilustración 116: Función de pérdidas DBOW

Al convertir los textos del corpus en vectores de longitud fija, obteniendo un vector por documento en lugar de un vector por palabra, este modelo apenas se ve influido por el sesgo de la longitud de las noticias.

7.5. Pruebas

Las pruebas realizadas en este sprint se centran principalmente en detectar posibles errores en la forma de crear definir los modelos implementados, además de comprobar que el funcionamiento de las funciones auxiliares implementadas (como lo son la división en conjunto de entrenamiento y pruebas), se realizan de la forma esperada.

P-05: División en conjunto de entrenamiento y pruebas	
Requisitos	REQ-03, REQ-04, REQ-05, REQ-06, REQ-07, REQ-08, REQ-09, REQ-10
Descripción	Realizar la división del dataset en conjunto de entrenamiento y conjunto de pruebas, con una distribución de un 70% y un 30% respectivamente
Pre-requisitos	Tener guardado el fichero nlp_functions.py en el directorio raíz del proyecto y el fichero ticnn_preprocessed.csv en la carpeta datasets
Entrada	Textos por clasificar y sus respectivas target variables como listas.
Salida esperada	Conjunto de entrenamiento y de pruebas en forma de lista, con una distribución del 70% y el 30%
Salida obtenida	Conjunto de entrenamiento y de pruebas en forma de lista, con una distribución del 70% y el 30%

Tabla 49: Prueba 5 - División en conjunto de entrenamiento y pruebas

El código de esta prueba se muestra en la Ilustración 117, y los resultados aparecen en la Ilustración 118.

```python
def prueba5():
    print('Prueba 5: División del dataset en conjunto de entrenamiento y pruebas')

    try:
        print('\n--- Lectura del dataset preprocesado')
        df = pd.read_csv('../dataset/ticnn_preprocessed.csv')
    except:
        print('Se ha producido un error al leer el dataset')
        raise

    print('Dataset leído correctamente, mostrando las tres primeras entradas')
    print(df.head(3))

    try:
        print('\n--- División de los datos en entrenamiento y pruebas')
        print('70% entrenamiento - 30% pruebas')

        texts = list(df['text'])
        targets = list(df['type'])

        x_train, y_train, x_test, y_test = nlp_f.train_test_split(texts, targets)
        print('Datos separados correctamente')

    except:
        print('Error al separar en entrenamiento y pruebas')
        raise

    try:
        print('\n--- Comprobamos que el tamaño de los conjuntos es correcto')

        assert len(x_train) == int(len(df)*0.7)
        assert len(x_train) == len(y_train)
        print('El tamaño del conjunto de entrenamiento es el correcto')

        assert len(x_test) == int(math.ceil(len(df)*0.3))
        assert len(x_test) == len(y_test)
        print('El tamaño del conjunto de pruebas es el correcto')

    except:
        print('El tamaño de los conjuntos no es el adecuado')
        raise

    print('\n----------------------------------------------------------------------')
    print('División en conjunto de entrenamiento y pruebas realizada correctamente')

```

Ilustración 117: Código Prueba 5

```
prueba5()
```

```
Prueba 5: División del dataset en conjunto de entrenamiento y pruebas

--- Lectura del dataset preprocesado
Dataset leído correctamente, mostrando las tres primeras entradas
                                                text  type
0  ['donald', 'trump', 'property', 'showcase', 'b...   0.0
1  ['trump', 'foundation', 'tell', 'new', 'york',...   0.0
2  ['donald', 'trump', 'prepares', 'white', 'hous...   0.0

--- División de los datos en entrenamiento y pruebas
70% entrenamiento - 30% pruebas
Datos separados correctamente

--- Comprobamos que el tamaño de los conjuntos es correcto
El tamaño del conjunto de entrenamiento es el correcto
El tamaño del conjunto de pruebas es el correcto

----------------------------------------------------------------------
División en conjunto de entrenamiento y pruebas realizada correctamente
```

Ilustración 118: Resultados de la Prueba 5

P-06: Convert to sparse tensor	
Requisitos	REQ-03, REQ-04, REQ-05
Descripción	Convertir un array de Numpy en un objeto de tipo SparseTensor de la librería TensorFlow que sirva como input para los modelos neuronales basados en Bag of Words.
Pre-requisitos	Tener guardado el fichero nlp_functions.py en el directorio raíz del proyecto.
Entrada	-
Salida esperada	Objeto de tipo SparseTensor
Salida obtenida	Objeto de tipo SparseTensor

Tabla 50: Prueba 6-Convert to sparse tensor

El código de esta prueba se muestra en la Ilustración 119, y los resultados aparecen en la Ilustración 120.

```
def prueba6():
    print('Prueba 6: Convertir un array de numpy a un sparse vector')

    try:
        print('\n--- Conversión a sparse tensor')

        test_vector = np.zeros(shape = (50000, ))
        test_vector_size = sys.getsizeof(test_vector)
        print('El tamaño de la matriz de numpy es {}'.format(test_vector_size))

        test_vector = scipy.sparse.csc_matrix(test_vector)

        sparse_vector = nlp_f.convert_to_sparse_tensor(test_vector)
        sparse_vector_size = sys.getsizeof(sparse_vector)

        print('El tamaño de la matriz sparse es {}'.format(sparse_vector_size))

        print('La matriz sparse es de tipo {}'.format(type(sparse_vector)))

        assert isinstance(sparse_vector,
                          type(tf.sparse.SparseTensor(indices=[[0, 0], [1, 2]], values=[1, 2], dense_shape=[3, 4])))

        assert test_vector_size > sparse_vector_size
        print('El tamaño del vector sparse es menor que el de la matriz de numpy')

    except:
        print('\nSe ha producido un error al intentar crear el vector sparse')
        raise

    print('\n----------------------------------------------------------------------')
    print('Tranformación a Matriz Sparse realizada correctamente')
```

Ilustración 119: Código Prueba 6

```
prueba6()

Prueba 6: Convertir un array de numpy a un sparse vector

--- Conversión a sparse tensor
El tamaño de la matriz de numpy es 400096
El tamaño de la matriz sparse es 56
La matriz sparse es de tipo <class 'tensorflow.python.framework.sparse_tensor.SparseTensor'>
El tamaño del vector sparse es menor que el de la matriz de numpy

----------------------------------------------------------------------
Tranformación a Matriz Sparse realizada correctamente
```

Ilustración 120: Resultados de la Prueba 6

P-07: Generar regresión logística	
Requisitos	REQ-03, REQ-04, REQ-05
Descripción	Creación de un modelo neuronal (Regresión logística)
Pre-requisitos	Tener guardado el fichero nlp_functions.py en el directorio raíz del proyecto.
Entrada	-
Salida esperada	Objeto Sequential de Tensorflow
Salida obtenida	Objeto Sequential de Tensorflow

Tabla 51: Prueba 7-Generar Regresión Logística

El código de esta prueba se muestra en la Ilustración 121.

```
1  def prueba7():
2      print('Prueba 7: Creación de un Modelo de Regresión logística')
3
4      try:
5          print('\nCreación del modelo...')
6          model = nlp_f.generate_logistic_regression(20000)
7
8      except:
9          print('Error al crear el modelo')
10         raise
11
12     print('\n-------------------------------------')
13     print('Modelo creado:')
14     print(model.summary())
15     print('\nClase del modelo: {}'.format(type(model)))
```

Ilustración 121: Código Prueba 7

Los resultados de esta prueba son los mostrados en la Ilustración 122.

```
prueba7()
Prueba 7: Creación de un Modelo de Regresión logística

Creación del modelo...

-----------------------------------
Modelo creado:
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (20, 1)                   20001
=================================================================
Total params: 20,001
Trainable params: 20,001
Non-trainable params: 0
_________________________________________________________________
None

Clase del modelo: <class 'tensorflow.python.keras.engine.sequential.Sequential'>
```

Ilustración 122: Resultados de la Prueba 7

P-08: Generar modelo neuronal Word2Vec	
Requisitos	REQ-06, REQ-07
Descripción	Creación de un modelo neuronal para clasificar los modelos basados en Word2Vec
Pre-requisitos	Tener guardado el fichero nlp_functions.py en el directorio raíz del proyecto.
Entrada	-
Salida esperada	Objeto Sequential de Tensorflow
Salida obtenida	Objeto Sequential de Tensorflow

Tabla 52: Prueba 8-Generar Modelo Neuronal Word2Vec

El código de esta prueba se muestra en la Ilustración 123.

```python
def prueba8():
    print('Prueba 8: Creación del Modelo Neuronal de Word2Vec')

    try:
        print('\nCreación del Modelo...')
        model = nlp_f.generate_w2v_dense_nn()

    except:
        print('Error al crear el modelo')
        raise

    print('\n-----------------------------------')
    print('Modelo creado:')
    print(model.summary())
    print('\nClase del modelo: {}'.format(type(model)))
```

Ilustración 123: Código Prueba 8

Los resultados de esta prueba son los mostrados en la Ilustración 124.

```
prueba8()
Prueba 8: Creación del Modelo Neuronal de Word2Vec

Creación del Modelo...

-----------------------------------
Modelo creado:
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 150)               45150
_________________________________________________________________
dropout (Dropout)            (None, 150)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 151
=================================================================
Total params: 45,301
Trainable params: 45,301
Non-trainable params: 0
_________________________________________________________________
None

Clase del modelo: <class 'tensorflow.python.keras.engine.sequential.Sequential'>
```

Ilustración 124: Resultados de la Prueba 8

P-09: Generar modelo neuronal Doc2Vec	
Requisitos	REQ-08, REQ-09
Descripción	Creación de un modelo neuronal para clasificar los modelos basados en Doc2Vec
Pre-requisitos	Tener guardado el fichero nlp_functions.py en el directorio raíz del proyecto.
Entrada	-
Salida esperada	Objeto Sequential de Tensorflow
Salida obtenida	Objeto Sequential de Tensorflow

Tabla 53: Generar Modelo Neuronal Doc2Vec

El código de esta prueba se muestra en la Ilustración 125.

```
def prueba9():
    print('Prueba 9: Creación del Modelo Neuronal de Doc2Vec')

    try:
        print('\nCreación del Modelo...')
        model = nlp_f.generate_d2v_dense_nn()

    except:
        print('Error al crear el modelo')
        raise

    print('\n-------------------------------------')
    print('Modelo creado:')
    print(model.summary())
    print('\nClase del modelo: {}'.format(type(model)))
```

Ilustración 125: Título Prueba 9

Los resultados de esta prueba son los mostrados en la Ilustración 126.

```
prueba9()
```

```
Prueba 9: Creación del Modelo Neuronal de Doc2Vec

Creación del Modelo...

-----------------------------------
Modelo creado:
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 150)               45150
_________________________________________________________________
dropout_1 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 151
=================================================================
Total params: 45,301
Trainable params: 45,301
Non-trainable params: 0
_________________________________________________________________
None

Clase del modelo: <class 'tensorflow.python.keras.engine.sequential.Sequential'>
```

Ilustración 126: Resultados de la Prueba 9

7.6. Resumen del Sprint

Como puede observarse en el Burndown de la Ilustración 127, no se han realizado avances hasta pasada una parte importante del Sprint, esto se debe a las labores de investigación realizadas para obtener conocimientos prácticos y teóricos sobre los métodos aplicados, además de aprender a usar TensorFlow 2.0 correctamente.

Los conocimientos adquiridos sobre TensorFlow y procesamiento del lenguaje natural también cubren muchos de los requisitos a satisfacer en futuros Sprints, por lo que la gestión del tiempo ha sido buena y se ha realizado un trabajo constante, aunque el burndown pueda sugerir lo contrario. Todos los objetivos del Sprint Backlog se han cumplido con éxito y dentro de los plazos establecidos, sin sufrir ningún retraso.

La consecución de este Sprint ha llevado un total de 98 horas, aunque 37 de ellas han sido dedicadas exclusivamente a labores de investigación. Un total de 61 horas no supone una desviación importante con respecto a las 60 que se estimaron en un inicio, especialmente si se tiene en cuenta que este sprint es el más exigente de todos.

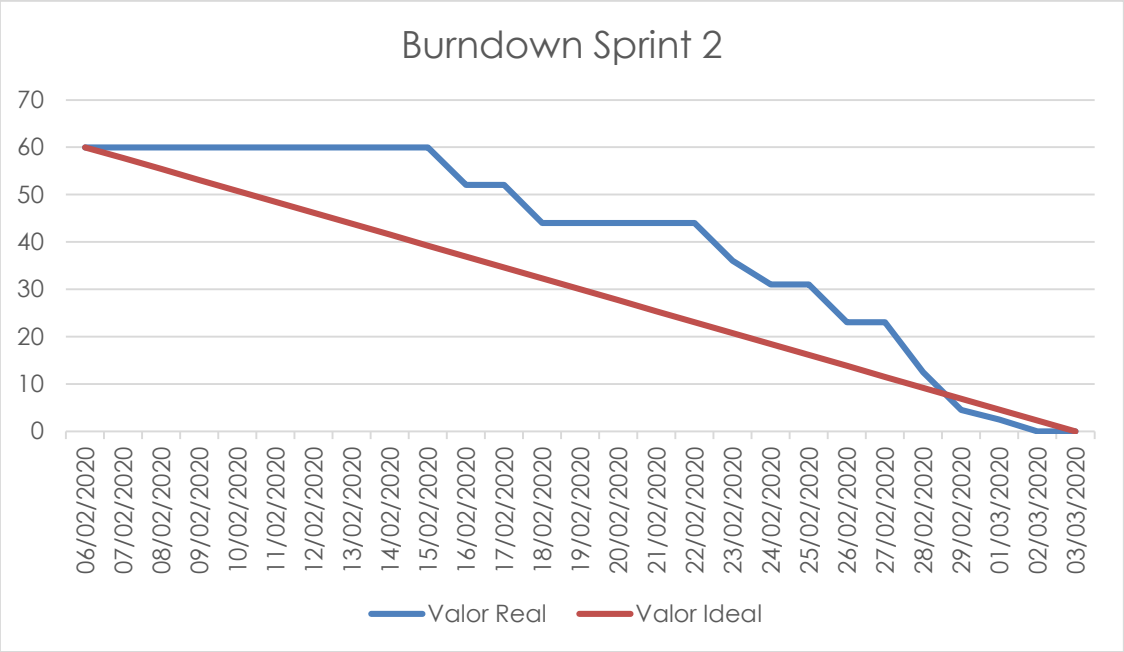


Ilustración 127: Burndown Sprint 2

8. Sprint 3: Implementación de Red Neuronal Recurrente

Durante este sprint se realizará la implementación del último modelo especificado en los requisitos, una Red Neuronal Recurrente de tipo Long Short Term Memory. Este sprint tiene una duración de 37 días, con un total de 60 horas asignadas.

8.1. Requisitos

El requisito por satisfacer en este sprint es el siguiente:

REQ-10: Red Neuronal Recurrente
Implementar una Red Neuronal Recurrente para la clasificación de los datos, siguiendo los principios de elección, entrenamiento, evaluación del modelo y ajuste de parámetros.

Tabla 54: REQ-10

Además de implementar esta Red, se realizará un estudio comparativo entre los distintos métodos implementados a lo largo del proyecto, analizando las ventajas e inconvenientes de cada uno de ellos.

8.2. Diseño

8.2.1. Diagrama de clases

El diagrama completo puede verse en la página siguiente (Ilustración 128). Los métodos de color rojo son los nuevos métodos que se implementarán durante el desarrollo de este sprint.

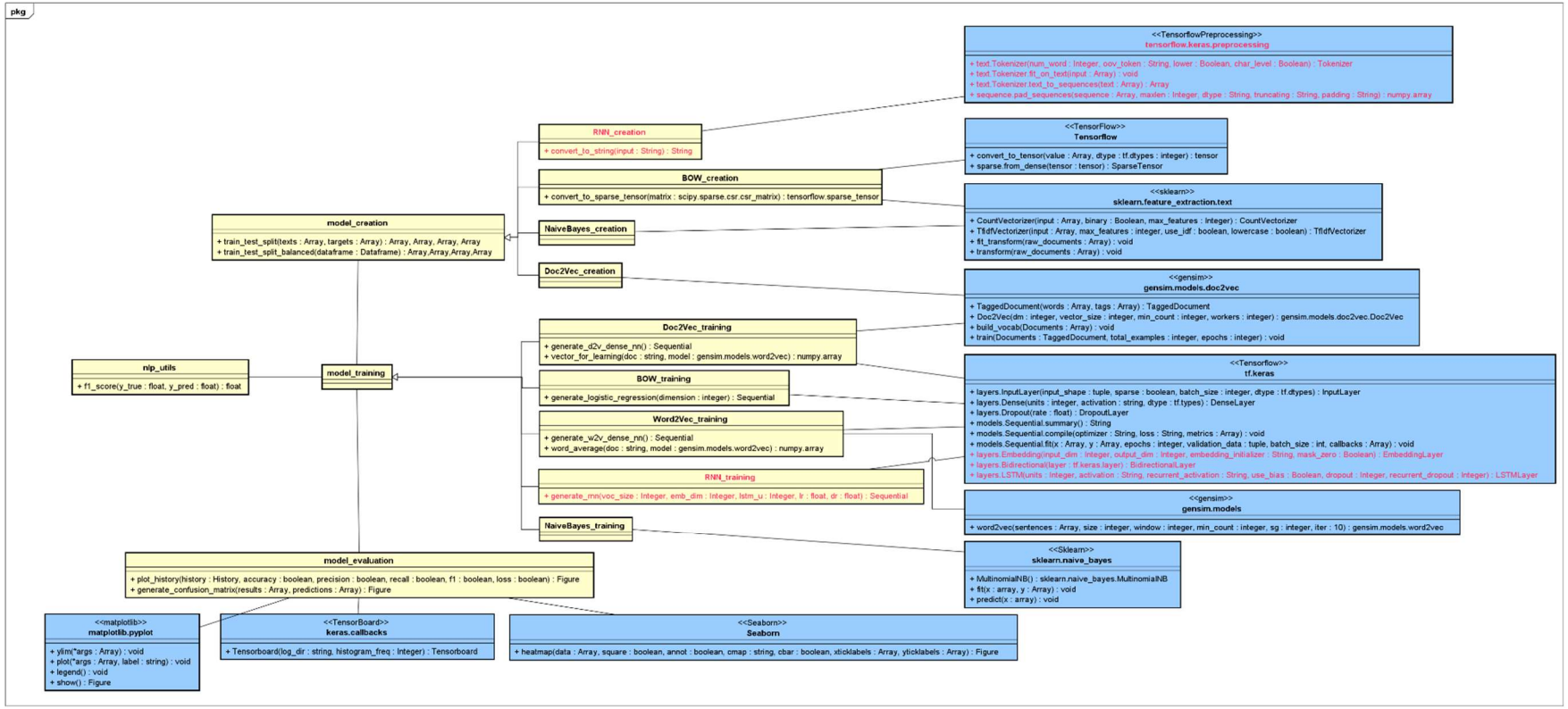


Ilustración 128: Diagrama de Clases a implementar en el Sprint 3

8.2.2. Red Neuronal Recurrente

La principal diferencia existente entre las Redes Neuronales Recurrentes y el perceptrón multicapa reside en que la Red Neuronal Recurrente puede realizar conexiones entre las unidades ocultas, lo que permite que estas puedan retener información en el tiempo a corto y largo plazo (la información persiste en el tiempo) [76].

Inicialmente, se consideró la opción de implementar una Red Neuronal Convolucional en lugar de una Red Neuronal Recurrente, ya que ambas estructuras muestran buenos resultados en la práctica y son los dos modelos más avanzados para realizar clasificación de textos (aunque las CNN suelen ser más comunes en clasificación de imágenes).

Los dos tipos de redes se caracterizan por tener en cuenta las secuencias de palabras. No obstante, las RNN tienen la ventaja de verse menos afectadas por las entradas de longitud variable que las CNN, por lo que, dado el sesgo descubierto en el Sprint 2, esta estructura parece ser la opción más adecuada.

Este tipo de redes contienen bucles, lo que permite que la información persista [77].

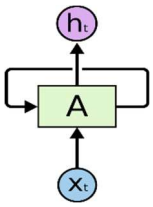


Ilustración 129: Red Neuronal Recurrente básica

Una red neuronal recurrente también puede entenderse como múltiples copias de una misma red:

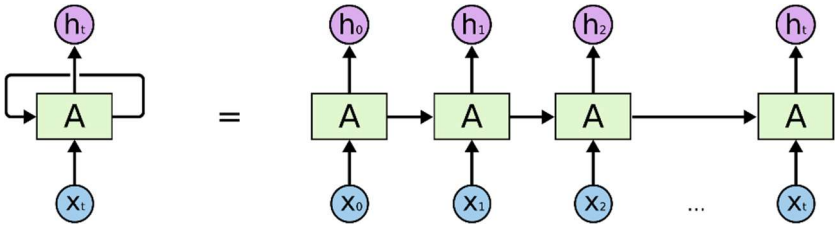


Ilustración 130: Red Neuronal Recurrente básica desenrrollada

Esta estructura se diseñó originalmente para trabajar con series temporales, pero en los últimos años han demostrado ofrecer resultados muy buenos en tareas como: Reconocimiento de voz, traducción, Language modelling, reconocimiento de imágenes y el caso que nos ocupa en este proyecto, clasificación de textos [77] [78].

Desafortunadamente, las Redes Neuronales Recurrentes tradicionales no pueden capturar dependencias a largo plazo, ya que se reescriben por completo pasadas varias entradas. Este problema se conoce como desvanecimiento del gradiente, y se soluciona empleando estructuras que mejoran el diseño de las RNN tradicionales para que tengan la capacidad de aprender dependencias a largo plazo, como es el caso de las redes Long Short Term Memory (LSTM) [79].

Las redes LSTM tienen la capacidad de añadir o eliminar información del estado de cada celda, emulando hasta cierto el punto el concepto de memoria a corto plazo del cerebro humano, este proceso se regula mediante estructuras conocidas como gates (o válvulas). Estas gates son de tres tipos:

- **Forget gate:** Determina que partes de la celda de memoria deben ser olvidadas al introducir un nuevo input.
- **Input gate:** Determina que partes del input actual deben escribirse en la celda de memoria.
- **Output gate:** Determina que partes de la celda de memoria deben ser volcadas en la memoria actual.

Cada una de las celdas de una red neuronal recurrente de tipo LSTM sigue la siguiente estructura:

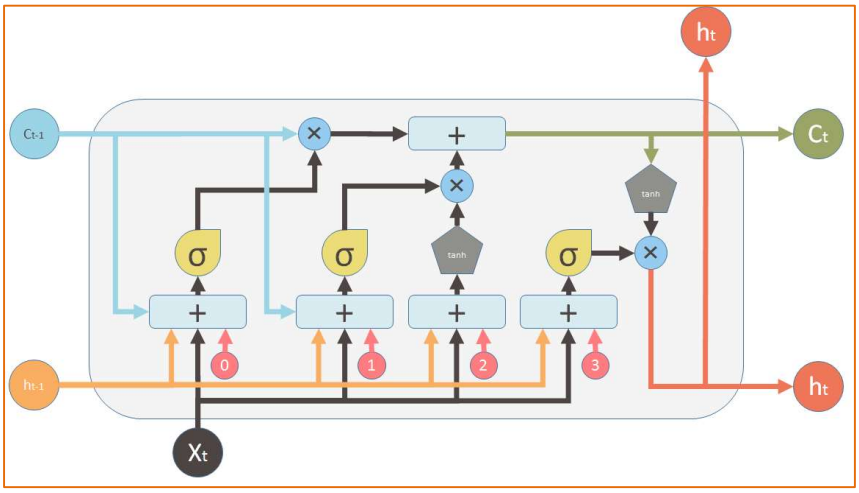


Ilustración 131: Celda LSTM [80]

La nomenclatura es la siguiente:

- x_t: Vector de entrada de la capa anterior (en nuestro caso la capa de embedding)
- c_{t-1}: Memoria de la celda anterior.
- c_t: Memoria de la celda actual.
- h_{t-1}: Salida (output) de la celda anterior.
- h_t: Salida de la celda actual.

De esta forma, cada unidad puede tomar decisiones teniendo en cuenta su input, salidas de celdas anteriores y su propia memoria, generando una nueva salida que altera su memoria. En las siguientes ilustraciones se muestran las gates de cada celda:

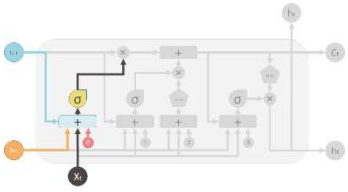


Ilustración 132: Forget Gate

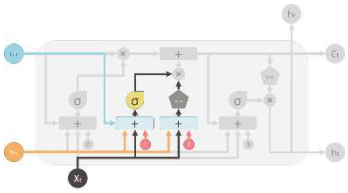


Ilustración 133: Input Gate

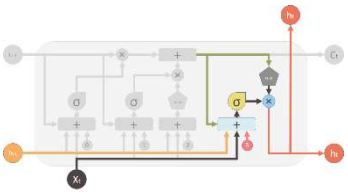


Ilustración 134: Output Gate

Las distintas celdas que componen la capa LSTM se unen como se muestra en la Ilustración 135:

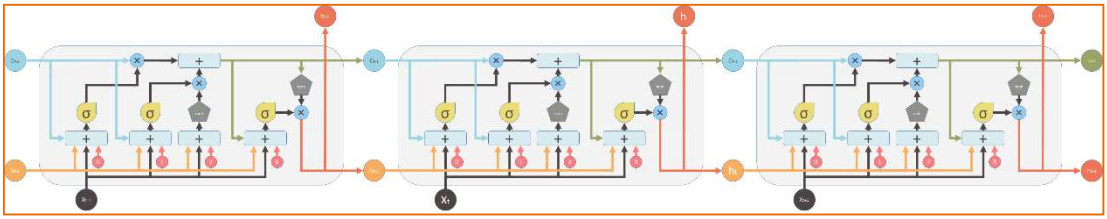


Ilustración 135: Conexión entre celdas LSTM [80]

A la hora de implementar la capa LSTM, utilizaremos una capa bidirrecional como wrapper. Las capas Bidireccionales suelen utilizarse en secuencias que no son series temporales, ya que las RNN ofrecen mejores resultados si en lugar de procesar las secuencias solo de principio a fin, también las procesamos al revés. Es muy común utilizarlo en textos para conocer el contexto de cada palabra (en caso contrario, solo podríamos conocer las palabras anteriores o posteriores).

Esta capa también nos permite hacer que toda la secuencia tenga igual importancia a la hora de realizar la clasificación, en caso de no emplear una capa bidireccional, el final de cada secuencia (en este caso, textos) sería más determinante que el principio, lo cual no tiene justificación la mayoría de las veces que se realiza la clasificación de un documento.

La estructura de la RNN a implementar está compuesta por las siguientes capas:

- **Capa de Embedding:** Convierte las secuencias en un output de dimensión fija.
- **Capa Bidireccional LSTM:** Esta capa utiliza la tangente hiperbólica como función de activación, y la función sigmoide como función de activación del paso recurrente.
- **Capa Fully Connected 1:** Con función de activación Relu.
- **Capa Dropout 1**
- **Capa Fully Connected 2:** Con función de activación Relu.
- **Capa Dropout 2**
- **Capa de Salida:** Perceptrón con función de activación sigmoide.

8.3. Implementación

En esta sección se desarrollan los detalles referentes a la implementación de la Red Neuronal Recurrente.

8.3.1. Creación del modelo

Al igual que ocurría en los modelos anteriores, comenzamos cargando el dataset preprocesado como objeto dataframe de pandas y utilizamos la función sample para alterar el orden aleatoriamente. Guardamos los documentos y sus respectivas target variables como listas, que se utilizan como parámetros de la función train_test_split.

```
[ ]  # Lectura del dataset
     df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/TFG/datasets/ticnn_preprocessed.csv')

     # Alteramos el orden aleatoriamente
     df = df.sample(frac=1)
     # Convertimos la columna content de string a lista de string
     from ast import literal_eval
     df['text'] = df['text'].apply(literal_eval)

     # Aplicamos la función convert_to_string
     df['text'] = df['text'].apply(convert_to_string)

     # Convertimos los textos y las targets variables a listas
     texts = list(df['text'])
     targets = list(df['type'])

     # Dividimos en conjunto de entrenamiento y conjunto de pruebas
     x_train, y_train, x_test, y_test = nlp_f.train_test_split(texts, targets)
```

Ilustración 136: Lectura del dataset

La principal diferencia en esta ocasión es que el tokenizador de TensorFlow no recibe como parámetro una lista de strings, sino las noticias como un texto, por lo que definimos la función convert_to_string, que convierte las listas de string al formato requerido:

```
[ ]  # Para tokenizar, debemos convertir las listas de string a string nuevamente
     def convert_to_string(input):
       res = ''
       for word in input:
         res = res + ' ' + word

       return res
```

Ilustración 137: Función convert_to_string

A continuación, llamamos a la función tokenizer de TensorFlow, que permite vectorizar un corpus convirtiendo cada texto de este en una secuencia de enteros, en el que cada entero se asocia con una palabra en un objeto de tipo diccionario. Los argumentos que recibe esta función son [81]:

- **num_words:** Tamaño del vocabulario, basado en la frecuencia de las palabras, solo se emplearán las num_words – 1 palabras más frecuentes.

- **oov_token:** Es el string que reemplazará a las palabras que se encuentren fuera del vocabulario por no aparecer con suficiente frecuencia.
- **char_level:** De tipo booleano, si vale true, cada carácter es tratado como un token.
- **lower:** De tipo booleano, si vale true, convierte el texto a minúsculas. En este caso no se aplica porque este paso ya se ha realizado durante el preprocesamiento del conjunto de datos.

Creamos el objeto de tipo tokenizer y llamamos a la función fit_on_text para que se adapte al vocabulario del conjunto de entrenamiento.

```
[ ]  tokenizer = tf_tokenizer(num_words = vocabulary_size, oov_token='<null_token>', lower = False, char_level = False)
     tokenizer.fit_on_texts(x_train)
     word_dict = tokenizer.word_index
```

Ilustración 138: Tokenizer de TensorFlow

Si imprimimos las diez primeras entradas del diccionario (Ilustración 139), observamos que la palabra más común del corpus es el token definido para las palabras que no aparecen con suficiente frecuencia, seguido de “trump”, “say” y “mr”.

```
[ ]  dict(list(word_dict.items())[0:10])

     {'<null_token>': 1,
      'clinton': 4,
      'mr': 8,
      'one': 9,
      'people': 10,
      'say': 3,
      'state': 6,
      'trump': 2,
      'u': 5,
      'would': 7}
```

Ilustración 139: Palabras más comunes del Corpus

Aplicamos la función text_to_sequences a los conjuntos de entrenamiento y pruebas, que convierte los artículos en listas de enteros definidas por el diccionario anterior.

```
[ ]  x_train_sequence = tokenizer.texts_to_sequences(x_train)
     x_test_sequence = tokenizer.texts_to_sequences(x_test)

[ ]  print(x_train_sequence[0])
     print(x_test_sequence[0])

     [3263, 97, 66, 21, 4, 40, 2, 126, 21, 4, 12, 10289, 256, 361, 279, 829, 40,
     [2548, 6537, 14, 703, 3, 2370, 145, 1924, 785, 104, 161, 49, 113, 3356]
```

Ilustración 140: Conversión de los textos a secuencias

Las noticias tokenizadas son de longitud variable, y el tamaño del input de la RNN debe ser fijo, por lo tanto, utilizamos la técnica del padding para rellenar con ceros las noticias que no lleguen al tamaño deseado y recortar aquellas que lo superen.

```
[ ]  train_padded = pad_sequences(x_train_sequence, maxlen = padded_sequence_len, dtype = 'int32', truncating = 'post', padding = 'post')

     test_padded = pad_sequences(x_test_sequence, maxlen = padded_sequence_len, dtype = 'int32', truncating = 'post', padding = 'post')
```

Ilustración 141: Aplicación de la función padding a las secuencias

8.3.2. Entrenamiento del modelo

Para crear el modelo que sigue la estructura definida en la sección Diseño, implementamos la función generate_rnn (Ilustración 142), que recibe como parámetros el tamaño del vocabulario, el número de dimensiones del Word Embedding, el número de unidades de la capa LSTM, el ratio de dropout y el learning rate.

La capa bidireccional actúa como un "envoltorio" (wrapper) [82] de la capa que se le pasa por parámetro, en este caso una capa LSTM (Long Short Term Memory). La capa LSTM recibe los siguientes parámetros [83]:

- **units:** Tamaño del output (Número de unidades internas).
- **activation:** Función de activación.
- **recurrent_activation:** Función de activación del paso recurrente.
- **use_bias:** De tipo booleano, si su valor es true, se utiliza un vector bias, que por defecto se inicializa a cero.
- **dropout:** Valor del dropout.
- **recurrent_dropout:** Dropout del paso recurrente.

```
def generate_rnn(voc_size, emb_dim, lstm_u, lr, dr):
  model = tf.keras.Sequential()
  # Capa de Embedding
  model.add(
    tf.keras.layers.Embedding(input_dim = vocabulary_size,
                              output_dim = embedding_dimension,
                              embeddings_initializer = 'uniform',
                              mask_zero = True))
  # Capa bidireccional de LSTM
  model.add(
      # Capa Bidireccional
      tf.keras.layers.Bidirectional(
          # Capa LSTM
          tf.keras.layers.LSTM(units = lstm_units,
                               activation = 'tanh',
                               recurrent_activation = 'sigmoid',
                               use_bias = True,
                               dropout = dr,
                               recurrent_dropout = 0.05
                               )))
  # Capa Densa número 1
  model.add(
      tf.keras.layers.Dense(
          units = 128,
          activation = 'relu'))
  # Capa Dropout número 1
  model.add(tf.keras.layers.Dropout(rate = dr))
  # Capa Densa número 2
  model.add(
      tf.keras.layers.Dense(
          units = 64,
          activation = 'relu'))
  # Capa Dropout número 2
  model.add(tf.keras.layers.Dropout(rate = dr))
  # Capa de salida
  model.add(
      tf.keras.layers.Dense(
          units = 1,
          activation = 'sigmoid'))
  # Definimos el optimizador
  rmsprop_optim = tf.keras.optimizer.RMSprop(learning_rate=lr)
  # Compilamos el modelo
  model.compile(optimizer=rmsprop_optim, loss='binary_crossentropy'
,metrics=['accuracy', 'Precision', 'Recall', nlp_f.f1_score])

  return model
```

Ilustración 142: Función generate_rnn

El modelo tiene un total de 2.275.73 variables a optimizar:

```
[ ]  model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 100)         2000000
_________________________________________________________________
bidirectional (Bidirectional (None, 256)               234496
_________________________________________________________________
dense (Dense)                (None, 128)               32896
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 2,275,713
Trainable params: 2,275,713
Non-trainable params: 0
_________________________________________________________________
```

Ilustración 143: Variables de la Red Neuronal Recurrente

Por último, entrenamos el modelo:

```
[ ]  history = model.fit(train_padded,
                         np.array(y_train),
                         epochs = n_epochs,
                         validation_data = (test_padded, np.array(y_test)),
                         callbacks = [tensorboard_callback],
                         batch_size = 100
                         )
```

Ilustración 144: Entrenamiento del Modelo

8.3.3. Evaluación del modelo

De nuevo, para la evaluación del modelo se utilizan la función plot_history y la librería Tensorboard. El grafo de computación sigue la estructura mostrada en la Ilustración 145, en la que podemos apreciar la capa de embedding, seguida de la capa bidireccional, que conecta por último con dos capas densas a las que se aplica dropout, en la última capa densa se utiliza un solo perceptrón para clasificar los datos:

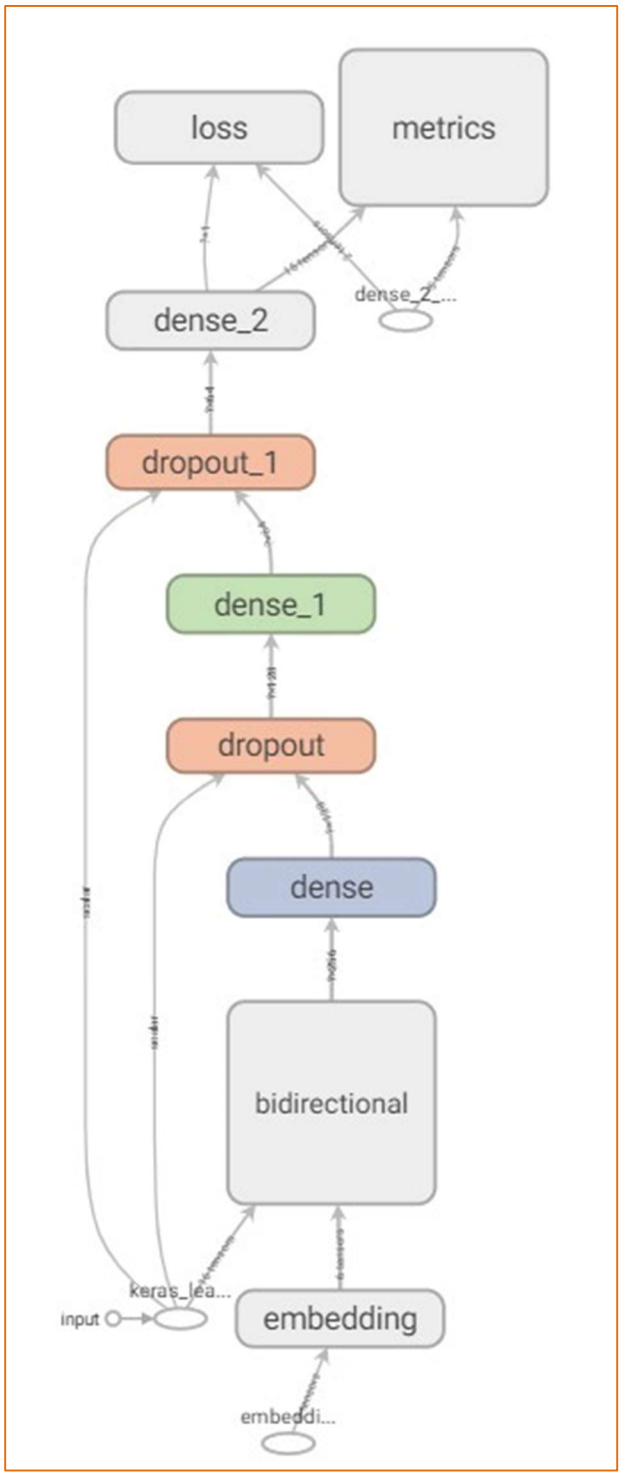


Ilustración 145: Grafo de computación de la RNN

8.4. Análisis de los resultados

Como se muestra en la Ilustración 146, a partir de la época 4 se produce un claro caso de Overfitting en la Red, por lo que se determina que el número máximo de épocas adecuado para entrenar el modelo es 3. Al contrario de lo que pueda parecer, este número de épocas no es inesperado, ya que este tipo de redes se caracteriza por su rápida convergencia.

Ilustración 146: Vectores de pérdidas RNN 10 épocas

En la Tabla 55 aparecen los hiperparámetros utilizados para entrenar la red neuronal.

Hiperparámetros	
Longitud Secuencias	600
Tamaño del Vocabulario	20000
Dimensiones de los Word Embeddings	100
Unidades LSTM	128
Épocas	3
Dropout	0.2
Learning Rate	0.001

Tabla 55: Hiperparámetros del modelo

En la Tabla 56 se muestran los resultados del modelo con los hiperparámetros de la tabla anterior, que alcanzan un F1-Score del 95%, con equilibrio entre precision y recall, lo que muestra que el modelo en ningún momento se está decantando por la clase mayoritaria.

	F1-Score	Recall	Precision	Accuracy	Loss
Entrenamiento	0.9949	0.9956	0.994	0.9939	0.0289
Pruebas	0.9576	0.962	0.9532	0.9507	0.1786

Tabla 56: Resultados RNN

Este modelo es, junto con Doc2Vec, el que ofrece mayor resistencia al sesgo de la longitud de las noticias, además de devolver resultados ligeramente mejores. También hay que tener en cuenta que, debido a su memoria a corto plazo y al análisis bidireccional de las noticias, la información aprendida persiste durante bastante más tiempo que en el caso de los otros

modelos, por lo que puede considerarse el modelo más robusto de todos. En el Sprint 4 se llevará a cabo un ajuste de sus hiperparámetros.

8.5. Comparación de los modelos implementados

En este apartado se realiza un estudio comparativo entre los distintos modelos que han sido implementados en los Sprints 2 y 3, analizando las ventajas e inconvenientes de cada uno de ellos. Es importante tener en mente que en Data Science nunca puede afirmarse que un modelo es directamente mejor que otro, ya que cada uno tiene sus aplicaciones y puede devolver mejores o peores resultados en función de determinados factores como son la distribución de los datos o el tipo de tarea a realizar.

El científico de datos es el encargado de escoger el mejor modelo en base a su intuición, experiencia previa y a un análisis del dataset. Aún con esto, existirán muchas ocasiones en las que la única forma de comprobar si un modelo es adecuado o no es implementándolo.

Además de realizar un análisis de los datos, también es recomendable implementar algunos métodos simples para detectar posibles sesgos o patrones en los datos, como ha sido el caso de Bag Of Words y Naive Bayes en este proyecto.

La Tabla 57 muestra, para cada uno de los modelos implementados, si determinados factores influyen en la clasificación realizada.

	Naive Bayes	BOW	Word2Vec	Doc2Vec	RNN
Orden de las palabras	No	No	No	Sí	Sí
Contexto de las palabras	No	No	Sí	Sí	Sí
Longitud de las Noticias	Sí	Sí	Sí	No	No

Tabla 57: Factores que influyen en cada tipo de modelo

La principal diferencia entre Naive Bayes y Bag Of Words reside en que el primero es un modelo probabilístico y el segundo es un modelo vectorial. Tal y como se expone en la sección Análisis de los resultados del Sprint 2, estos dos modelos son los que más afectados se ven por el sesgo de la longitud de las noticias, especialmente Bag Of Words que, en datasets como este, se ayudan del nivel de dispersión de los vectores para realizar la clasificación. Esto, unido al sesgo provocado por el vocabulario de las elecciones, nos permite explicar en gran medida los buenos resultados de los artículos en los que ha sido empleado este dataset.

La detección de estos sesgos gracias a la implementación y análisis de estos métodos nos ha permitido comprender mejor el conjunto de datos, escogiendo modelos que se vean menos afectados por ellos.

En Word2Vec obtenemos una representación vectorial densa de tamaño fijo de cada palabra en la que se tiene en cuenta el contexto de las mismas, sin embargo, la longitud de las noticias sigue teniendo cierta influencia en la clasificación, esto se debe a que la entrada de la red neuronal es la media aritmética de todas las palabras de cada noticia, por lo que, cuantas más palabras contenga la noticia, se realizará una media entre más elementos, lo que contribuye a la dispersión del valor de la media, por lo que este sesgo sigue presente, aunque en una medida mucho menor.

Este modelo es superior a los dos anteriores al verse menos afectado por la longitud de las noticas y tener en cuenta el contexto de las palabras, pero sigue ignorando el orden de aparición de estas.

Doc2Vec y la Red Neuronal Recurrente son los modelos que menos afectados se ven por la longitud de las noticias, al convertir cada documento en un vector de tamaño fijo. En este aspecto, Doc2Vec podría considerarse ligeramente superior al convertir el texto completo en un vector, sin necesidad de paddear o recortar los textos que no se adapten a una longitud específica.

No obstante, la longitud máxima seleccionada como máximo para cada noticia en la RNN apenas recorta un pequeño porcentaje de los textos que componen el corpus, por lo que la posible pérdida de información es muy leve.

La RNN compensa esta pequeña desventaja por mucho gracias a dos factores: Su memoria a corto plazo (LSTM, que le permite mantener la información durante más tiempo sin reescribirse) y la capa Bidireccional, gracias a la cual las secuencias pueden analizarse en ambas direcciones (de principio a fin y viceversa), ofreciendo mucha más información sobre el contexto de cada palabra. La memoria a corto plazo también resultaría útil a la RNN en el caso de que fuera desplegada y entrenada con nuevos datos de manera constante, ya que no “olvidaría” los resultados antiguos, lo que permitiría a la red adaptarse a nuevos ámbitos informativos.

Por estos motivos llegamos a la conclusión de que la RNN es el modelo más adecuado para realizar esta tarea de clasificación con los datos de los que disponemos.

8.6. Pruebas

P-10: Generar Red Neuronal Recurrente	
Requisitos	REQ-10
Descripción	Creación de una Red Neuronal Recurrente
Pre-requisitos	Tener guardado el fichero nlp_functions.py en el directorio raíz del proyecto.
Entrada	-
Salida esperada	Objeto Sequential de Tensorflow
Salida obtenida	Objeto Sequential de Tensorflow

Tabla 58: Prueba 10-Generar Red Neuronal Recurrente

El código de esta prueba se muestra en la Ilustración 147

```
1  def prueba10():
2      print('Prueba 10: Creación de la Red Neuronal Recurrente')
3
4      try:
5          print('\nCreación del Modelo...')
6          model = nlp_f.generate_rnn(20000, 100, 128)
7
8      except:
9          print('Error al crear el modelo')
10         raise
11
12     print('\n---------------------------------------')
13     print('Modelo creado con éxito')
14     print(model.summary())
15     print('\nClase del modelo: {}'.format(type(model)))
```

Ilustración 147: Creación de la RNN

Los resultados de esta prueba son los mostrados en la Ilustración 148

```
prueba10()
```

```
Prueba 10: Creación de la Red Neuronal Recurrente

Creación del Modelo...

--------------------------------------
Modelo creado con éxito
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 100)         2000000
_________________________________________________________________
bidirectional (Bidirectional (None, 256)               234496
_________________________________________________________________
dense (Dense)                (None, 128)               32896
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 2,275,713
Trainable params: 2,275,713
Non-trainable params: 0
_________________________________________________________________
None

Clase del modelo: <class 'tensorflow.python.keras.engine.sequential.Sequential'>
```

Ilustración 148: Resultado de la Prueba 10

8.7. Resumen del Sprint

La consecución de este sprint ha llevado un total de 54 horas, por lo que las horas trabajadas han sido cercanas a las estimadas.

Durante este sprint se ha realizado la implementación de la Red Neuronal Recurrente, el método de clasificación más avanzado y el que ofrece mejores resultados, además de ser el que se ve menos afectado por los sesgos detectados en el sprint 2 gracias a los baselines aplicados, por lo que se trata del modelo que mejor aprovecha el conjunto de datos en el que ha sido entrenado.

En la Ilustración 149 se muestra el gráfico Burndown del sprint, que refleja un ritmo de trabajo constante y continuo, todas las tareas se han cumplido dentro de los plazos establecidos.

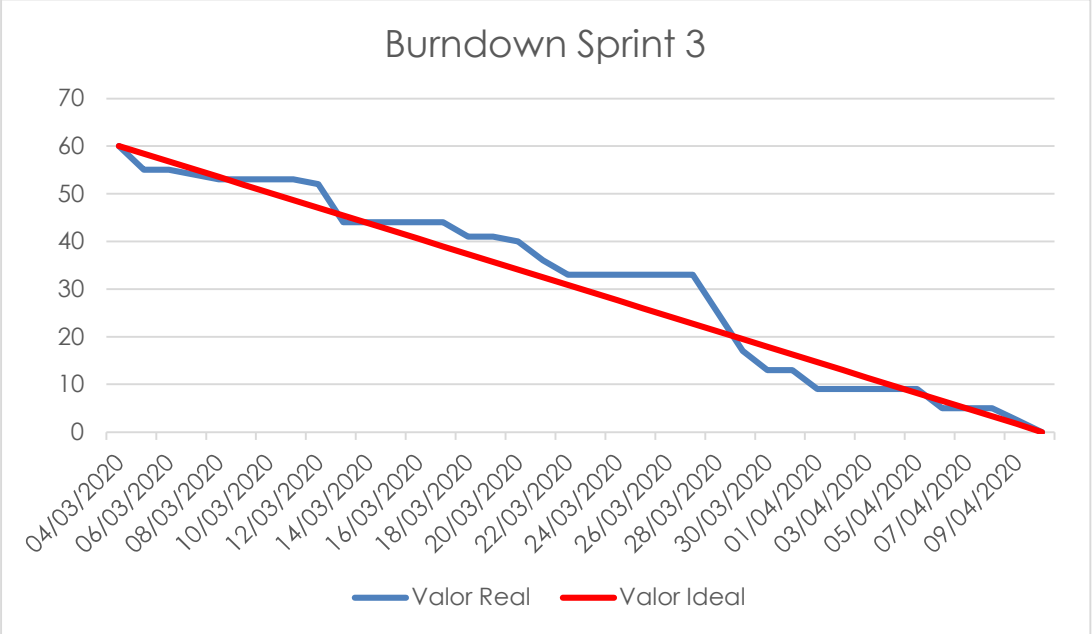


Ilustración 149: Burndown Sprint 3

9. Sprint 4: Optimización de Hiperparámetros

9.1. Requisitos

Durante este Sprint se realizará la optimización de hiperparámetros de la Red Neuronal Recurrente implementada a lo largo del Sprint 3, con el objetivo de mejorar los resultados de la esta, aprovechando al máximo la capacidad del modelo. Por tanto, el requisito a satisfacer para este incremento es el siguiente:

REQ-11: Optimización de Hiperparámetros
Optimizar los hiperparámetros de la Red Neuronal Recurrente

Tabla 59: REQ-11

Además de esto, también se reestructurará el código del proyecto, hasta ahora implementado en notebooks de Jupyter o Google Colab, para que siga la estructura indicada en los diagramas de clases.

Por último, se redactarán los manuales de uso e instalación; las desviaciones de tiempo y costes; y las conclusiones del proyecto.

9.2. Diseño

A la hora de crear modelos en el mundo del Machine Learning, es necesario seleccionar varios hiperparámetros (variables cuyo valor se establece antes del entrenamiento), como el ratio de Dropout y el learning rate. La elección de sus valores tiene repercusiones directas en las métricas que evalúan el rendimiento del modelo.

Por lo tanto, un paso imprescindible en el proceso de entrenamiento de un modelo consiste en identificar los mejores hiperparámetros para el problema, que deben obtenerse mediante experimentación. Este proceso se conoce como Optimización de Hiperparámetros o, en inglés, “Hyperparameter Tuning”.

Para realizar la optimización, la técnica a emplear es grid search o búsqueda en cuadrícula, que consiste en realizar una búsqueda exhaustiva a través de un subconjunto de hiperparámetros seleccionados. La métrica utilizada para diferenciar las diferentes configuraciones del modelo será F1-Score.

El subconjunto de hiperparámetros seleccionados para la búsqueda y los valores seleccionados son los mostrados en la Tabla 60:

Hiperparámetro	Valores
Ratio de Dropout	{0.2, 0.4}
Número de Unidades LSTM	{128, 256}
Batch Size	{50, 100, 200}
Learning Rate	{0.001, 0.01, 0.05}

Tabla 60: Hiperparámetros a Optimizar

Para la implementación se utilizará la API de síntesis de Hiperparámetros de TensorBoard. El diagrama de clases de este Sprint es el siguiente:

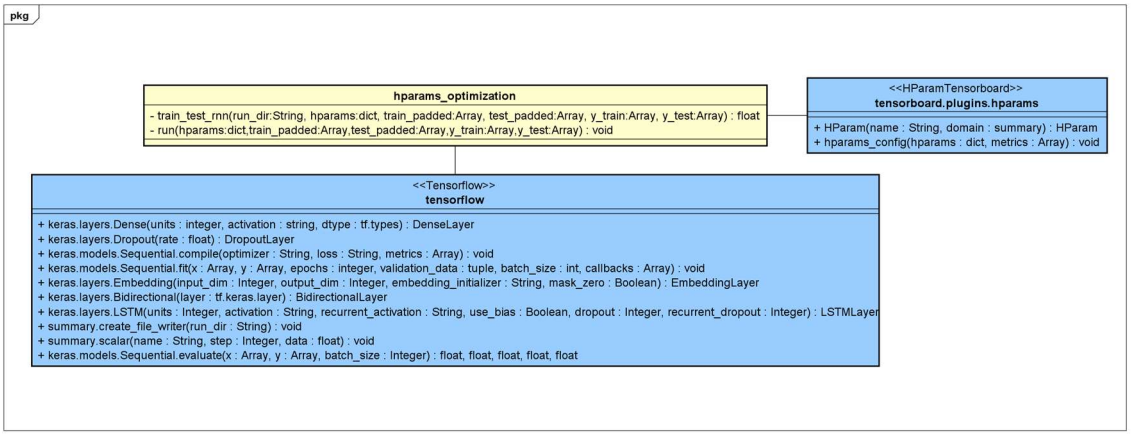


Ilustración 150: Diagrama de Clases Sprint 4

9.3. Implementación

9.3.1. Optimización de hiperparámetros

Para realizar la implementación de grid search, comenzamos ejecutando todos los pasos de configuración del conjunto de datos realizados durante el sprint 3 (lectura del dataset, alteración aleatoria del orden, división en conjunto de entrenamiento y pruebas, tokenización, conversión en secuencias y padding), para tener las entradas de las distintas configuraciones del modelo cargadas en memoria.

Tras cargar TensorBoard y eliminar los logs de ejecuciones anteriores, definimos los hiperparámetros a optimizar y sus distintos valores, así como la métrica a emplear para comparar las configuraciones (Ilustración 151). Estos hiperparámetros y valores son los establecidos en la Tabla 60 de la sección Diseño de este sprint.

Configuración de los hiperparámetros

Listamos los hiperparámetros a optimizar y sus valores

```
# Importamos el plugin HParams de Tensorboard
from tensorboard.plugins.hparams import api as hp

# Listamos los parámetros a optimizar y sus distintos valores
HP_DROPOUT = hp.HParam('dropout', hp.RealInterval(0.2, 0.4))
HP_LR = hp.HParam('learning_rate', hp.Discrete([0.001, 0.01, 0.05]))
HP_BATCH = hp.HParam('batch_size', hp.Discrete([50, 100, 200]))
HP_LSTM = hp.HParam('lstm_units', hp.Discrete([128, 256]))

METRIC_F1 = 'f1-score'    ##############################
```

Ilustración 151: Configuración de los hiperparámetros

Establecemos el directorio “logs/hparam_tuning” como la ubicación en la que se almacenarán los logs de las distintas ejecuciones y que serán interpretadas por TensorBoard (Ilustración 152) .

```
with tf.summary.create_file_writer('logs/hparam_tuning').as_default():
    hp.hparams_config(
    hparams = [HP_DROPOUT, HP_LR, HP_BATCH, HP_LSTM],
    metrics = [hp.Metric(METRIC_F1, display_name = 'F1-Score')]
    )
```

Ilustración 152: Selección del directorio de logs

Antes de comenzar la ejecución de las configuraciones, definimos las funciones train_test_rnn (Ilustración 153) y run (Ilustración 154). La función train_test_rnn recibe como entrada los valores de los hiperparámetros de una configuración determinada y crea y entrena un modelo con la misma estructura que la Red Neuronal Recurrente del Sprint 3, aplicando dicha configuración. Una vez entrenado el modelo, lo evalúa utilizando el conjunto de pruebas. La función devuelve el valor de F1-Score en el conjunto de pruebas.

```python
def train_test_rnn(hparams):
    # ----Comenzamos definiendo el modelo
    model = tf.keras.Sequential()
    # Capa de Embedding
    model.add(tf.keras.layers.Embedding(input_dim = 20000, # Tamaño del vocabulario
                                        output_dim = 100, # Número de dimensiones de WE
                                        embeddings_initializer = 'uniform',
                                        mask_zero = True))
    # Capa bidireccional
    model.add(tf.keras.layers.Bidirectional(
        # Capa LSTM
        tf.keras.layers.LSTM(units = hparams[HP_LSTM],   # Hiperparámetro a optimizar
                             activation = 'tanh',
                             recurrent_activation = 'sigmoid',
                             use_bias = True,
                             dropout = hparams[HP_DROPOUT],   # Hiperparámetro a optimizar
                             recurrent_dropout = 0.05)))
    # Capa densa 1
    model.add(tf.keras.layers.Dense(units = 128, activation = 'relu'))

    # Capa Dropout 1
    model.add(tf.keras.layers.Dropout(rate = hparams[HP_DROPOUT]))  # Hiperparámetro a optimizar

    # Capa densa 2
    model.add(tf.keras.layers.Dense(units = 64, activation = 'relu'))

    # Capa Dropout 2
    model.add(tf.keras.layers.Dropout(rate = hparams[HP_DROPOUT]))  # Hiperparámetro a optimizar

    # Capa de salida
    model.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))

    # Definimos el optimizador
    rmsprop_optim = tf.keras.optimizers.RMSprop(learning_rate = hparams[HP_LR])  # Hiperparámetro

    # ----Compilamos el modelo
    model.compile(optimizer= rmsprop_optim, loss = 'binary_crossentropy',
                  metrics = ['accuracy', 'Precision', 'Recall', nlp_f.f1_score])

    # ----Entrenar el modelo (No es necesario llamar a tensorboard)
    model.fit(train_padded, np.array(y_train), epochs = 3,
              batch_size = hparams[HP_BATCH])    # Hiperparámetro a optimizar

    # ----Evaluar el conjunto de pruebas
    # loss, acc, pre, rec, f1
    _, _, _, _, f1 = model.evaluate(test_padded, np.array(y_test))


    return f1
```

Ilustración 153: Función train_test_rnn

La función run recibe como parámetro una configuración de hiperparámetros y el nombre de un directorio. Esta función se encarga de llamar a train_test_rnn para obtener el valor de F1-Score de dicha configuración. Por último, genera los logs con los resultados y los guarda en el directorio indicado para que puedan ser interpretados por TensorBoard.

```python
def run(run_dir, hparams):
    with tf.summary.create_file_writer(run_dir).as_default():
        hp.hparams(hparams)   # Almacenar los valores de los parámetros usados
        f1 = train_test_rnn(hparams)  # Entrenamos el modelo con los valores especificados

        tf.summary.scalar(METRIC_F1, f1, step = 1)
```

Ilustración 154: Función run

Por último, solamente nos falta iterar sobre todas las posibles configuraciones, este paso es el más exigente en tiempo y computación de todo el proyecto tomando más de 25 horas para completarse. Esta ejecución ha tenido que realizarse en un cuaderno de Jupyter en lugar de en Google Colaboratory, ya que este último desconecta el entorno de ejecución de forma automática tras cierto tiempo.

Ejecutar este proceso en un solo bloque puede ser problemático porque cualquier fallo o interrupción inesperada puede suponer la pérdida de todo el progreso y los logs generados, algo que resulta inaceptable cuando se trabaja con limitaciones de tiempo, además de suponer un importante desperdicio de recursos en problemas del mundo real. Por estos motivos, se ha dividido la ejecución en cuatro partes como la mostrada en la Ilustración 155. En cada iteración, se actualizan los hiperparámetros y se llama a la función run, además de imprimir por pantalla el progreso.

```
session_num = 0
dropout_rate = HP_DROPOUT.domain.min_value
lstm_u = HP_LSTM.domain.values[0]

for learning_rate in HP_LR.domain.values:
    for batch_size in HP_BATCH.domain.values:
        hparams = {
            HP_DROPOUT: dropout_rate,
            HP_LR: learning_rate,
            HP_BATCH: batch_size,
            HP_LSTM: lstm_u
        }

        run_name = 'run-%d' % session_num
        print('--Iniciando ejecución : %s' % run_name)
        print({h.name: hparams[h] for h in hparams})
        run('logs/hparam_tuning/' + run_name, hparams)
        session_num += 1
```

Ilustración 155: Ejecución de Grid Search

Una vez generados los logs podemos cargar TensorBoard para comparar las distintas configuraciones del modelo. El panel de TensorBoard para análisis de hiperparámetros está compuesto por tres pestañas:

- **Table View (Ilustración 158):** Lista las distintas configuraciones, con los valores elegidos para los hiperparámetros y los resultados obtenidos.
- **Parallel Coordinates View (Ilustración 156):** Muestra cada ejecución como una línea que atraviesa un eje por cada hiperparámetro y métrica lo que permite identificar que grupos de hiperparámetros son más importantes.
- **Scatter plot matrix View (Ilustración 157):** Muestra gráficas representando la relación hiperparámetro/métrica, lo que ayuda a identificar correlaciones.

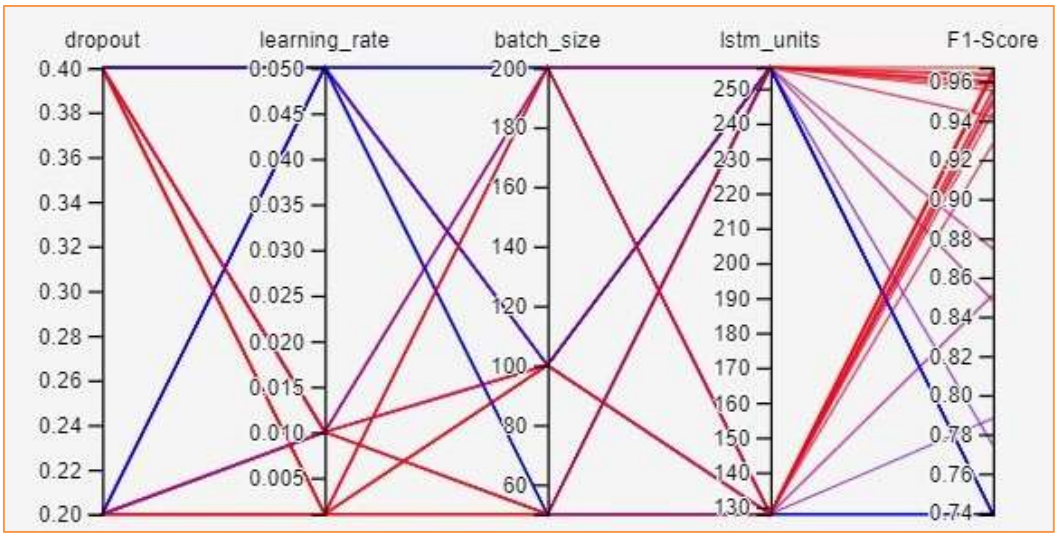


Ilustración 156: Parallel Coordinates View

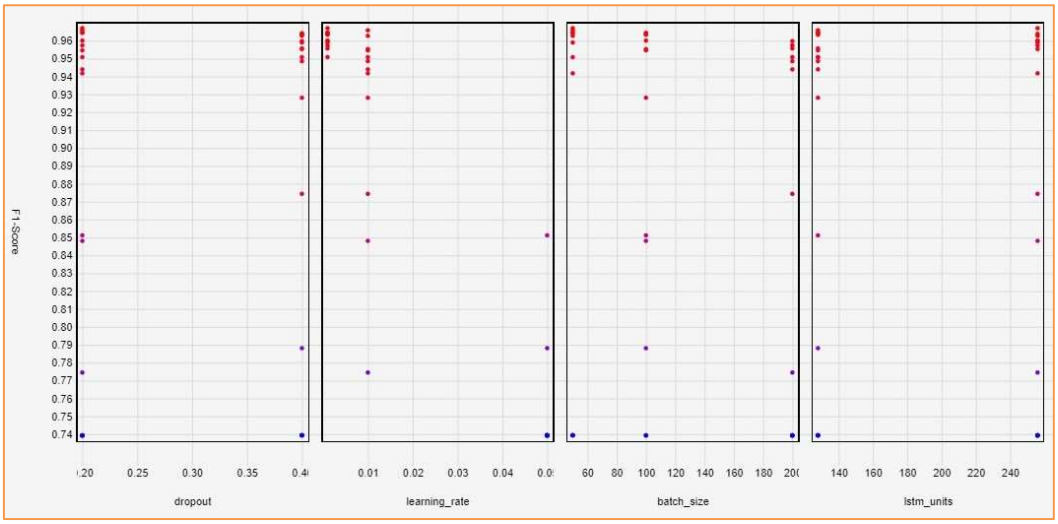


Ilustración 157: Scatter Plot View

Ayudándonos del filtrado por hiperparámetros que ofrece TensorBoard, podemos analizar la influencia del valor de cada uno de ellos en los resultados finales de forma aislada, permitiendo descartar rápidamente aquellos valores que ofrecen malos resultados independientemente de los valores del resto de hiperparámetros.

El ejemplo más claro de esto se da en las configuraciones en las que el learning rate vale 0.05, todos los modelos con este valor rondan entre el 73% y el 85% de F1-Score. Observando el valor de precision y recall de esas configuraciones, nos damos cuenta de que en muchas de ellas la red se está limitando a predecir la clase mayoritaria, por lo que podemos descartar directamente todas las ejecuciones que tengan este valor en el learning rate, ya que es insuficiente para garantizar una correcta convergencia de los modelos.

Una vez eliminadas aquellas configuraciones en las que el learning rate vale 0.05, el resto de las variantes ofrecen resultados bastante buenos, superando en la mayoría de los casos el 90% de

F1-Score. Todavía quedan algunos casos en los que el valor de F1-Score oscila entre el 77% y el 87%, pero estos ya no se limitan a predecir la clase mayoritaria.

El scatter plot (Ilustración 157) nos muestra que todos los modelos con learning rate = 0.001 consiguen resultados en torno al 95% de F1-Score, por lo que podemos considerar este hiperparámetro como el que más influye en los resultados finales del modelo. Muchas de las configuraciones con learning rate = 0.01 también alcanzan valores similares, pero en uno de cada tres ocasiones no lo hacen, por esta razón se ha escogido 0.001 como la mejor opción para el learning rate.

La diferencia en el rendimiento de las configuraciones restantes no es muy notoria (Ilustración 158), y factores como la distribución inicial de los pesos de la red y la división en entrenamiento y pruebas pueden hacer que en un nuevo entrenamiento los resultados sean ligeramente mejores en otra de estas 12 configuraciones, por lo que podemos considerarlas todas como válidas (es importante destacar que todas estas configuraciones han sido entrenadas con exactamente la misma división en entrenamiento y pruebas).

Trial ID	Show Metrics	dropout	learning_rate	batch_size	lstm_units	F1-Score
22d98bdd324ee0f...	☐	0.20000	0.0010000	200.00	256.00	0.95736
27bd004607111e...	☐	0.40000	0.0010000	50.000	128.00	0.96416
3137962589b916...	☐	0.20000	0.0010000	100.00	128.00	0.96435
3c87d2a0f326e6d...	☐	0.40000	0.0010000	50.000	256.00	0.95895
6adde88cc698079...	☐	0.20000	0.0010000	50.000	256.00	0.96696
79a357bc76980a...	☐	0.40000	0.0010000	200.00	128.00	0.95565
99db9fa451d3b1b...	☐	0.40000	0.0010000	200.00	256.00	0.95973
9b972386668d3c...	☐	0.40000	0.0010000	100.00	256.00	0.96380
ca7d3bc2f5cbb8f...	☐	0.20000	0.0010000	100.00	256.00	0.96003
e2981d7e0ab63c3...	☐	0.20000	0.0010000	50.000	128.00	0.96449
e3fcd27d2242ded...	☐	0.40000	0.0010000	100.00	128.00	0.96329
fe47438ab5e7165...	☐	0.20000	0.0010000	200.00	128.00	0.95080

Ilustración 158: Table View de las configuraciones con learning rate = 0.001

Como configuración definitiva del modelo, nos quedamos con aquella que ha ofrecido mejores resultados, que es la mostrada en la Tabla 61.

Configuración definitiva del modelo	
Ratio de Dropout	0.2
Learning Rate	0.001
Batch Size	50
Unidades LSTM	256
F1-Score	0.96696

Tabla 61: Configuración definitiva del modelo

9.3.2. Estructuración del código desarrollado

Durante el desarrollo del proyecto, todo el código implementado ha sido escrito en notebooks de Jupyter o Google Colab, lo que permite representarlo de forma más sencilla, gracias a la realización de representaciones gráficas y la capacidad de almacenar las variables entre celdas.

Las funciones definidas hasta ahora se encuentran o bien en los notebooks o en el archivo "nlp_functions.py", por lo que sus relaciones no estaban patentes ni resultaban lo bastante claras para alguien ajeno al desarrollo del proyecto. Para solucionar esto y que la ejecución del proyecto sea lo más limpia y estructurada posible, se han trasladado todas las funciones desarrolladas en los notebooks a ficheros .py de Python. Estos ficheros siguen la estructura del diagrama de clases del proyecto.

Las funciones incluidas dentro de cada fichero son las siguientes:

- *model_creation.py*
  - *train_test_split(texts, targets)*
  - *train_test_split_balanced(dataframe)*
  - *convert_to_sparse_tensor(matrix)*
  - *convert_to_string(input)*
- *model_training.py*
  - *generate_logistic_regression(dimension)*
  - *generate_w2v_dense_nn()*
  - *generate_d2v_dense_nn()*
  - *generate_rnn(voc_size, embd_dim, lstm_u, lr, dr)*
  - *word_average(doc, model)*
  - *vector_for_learning(model, docs)*
- *nlp_utils.py*
  - *f1_score(y_true, y_pred)*
- *preprocessing.py*
  - *tokenizer(example_sent)*
  - *delete_no_title(title)*
  - *filter_dataset(dataframe)*
- *model_evaluation.py*
  - *plot_history(history, accuracy, precision, recall, f1, loss)*
  - *generate_confusion_matrix(results, predictions)*
- *tests.py*
  - *prueba1()*
  - *prueba2()*
  - *prueba3()*
  - *prueba4()*
  - *prueba5()*
  - *prueba6()*
  - *prueba7()*

    - *prueba8()*
    - *prueba9()*
    - *prueba10()*
    - *prueba11()*
- *hparams_optimization.py*
    - *run(run_dir, hparams, train_padded, test_padded, y_train, y_test)*
    - *train_test_rnn(hparams, train_padded, test_padded, y_train, y_test)*

El funcionamiento de cada una de estas funciones ha sido descrito en su correspondiente sprint.

También se han creado varios ficheros main para ejecutar cada uno de los modelos, así como el preprocesado y las pruebas (*main_test.py*, *main_preprocessing.py*, *main_bow.py*, *main_naive_bayes.py*, *main_w2v.py*, *main_d2v.py*, *main_rnn.py*).

9.4. Pruebas

P-11: Grid Search con un subset de los hiperparámetros	
Requisitos	REQ-11
Descripción	Realizar la optimización de hiperparámetros empleando un subconjunto de la configuración empleada en la optimización real
Pre-requisitos	Tener guardado el fichero nlp_functions.py en el directorio raíz del proyecto y el conjunto de datos preprocesado (ticnn_preprocessed.csv) en la carpeta datasets.
Entrada	-
Salida esperada	Métricas de cada modelo aplicado
Salida obtenida	Métricas de cada modelo aplicado

Tabla 62: Prueba 11-Grid Search con un subset de los hiperparámetros

Esta prueba puede parecer trivial en un principio, pero resulta de vital importancia para este Sprint. Probar todas las configuraciones de hiperparámetros es una tarea que consume muchos recursos temporales y computacionales. Por estas razones, resulta lógico aplicar primero Grid Search empleando solamente un subconjunto de estas combinaciones, para detectar posibles errores de compilación. El código de esta prueba es el mostrado en las Ilustraciones 159 y 160.

```
def prueba11():
    print('Prueba 11: Grid Search con un subset de los hiperparámetros')

    try:
        print('\n--- Lectura del dataset preprocesado')
        df = pd.read_csv('../dataset/ticnn_preprocessed.csv')
    except:
        print('Se ha producido un error al leer el dataset')
        raise

    print('Dataset Leído correctamente')

    try:
        print('\n--- División de los datos en entrenamiento y pruebas')
        # Alteración del orden
        df = df.sample(frac = 1.)
        # Transformamos la columna text a lista de string
        from ast import literal_eval
        df['text'] = df['text'].apply(literal_eval)

        # Aplicamos la función convert_to_string a los textos
        df['text'] = df['text'].apply(convert_to_string)

        # Convertimos los textos y las targets variables a listas
        texts = list(df['text'])
        targets = list(df['type'])

        # División en conjunto de entrenamiento y test
        x_train, y_train, x_test, y_test = nlp_f.train_test_split(texts, targets)

    except:
        print('Error al separar en entrenamiento y pruebas')
        raise

    print('Datos separados correctamente')
```

Ilustración 159: Código Prueba 11 - Parte 1

```
    try:
        print('\n--- Creación del diccionario (Vocabulario)')
        tokenizer = tf_tokenizer(num_words = 20000, oov_token = '<null_token>',
                                 lower = False, char_level = False)
        tokenizer.fit_on_texts(x_train)

    except:
        print('Error al crear el vocabulario')
        raise

    print('Vocabulario creado correctamente')

    try:
        print('\n--- Transformar los textos en secuencias y padding')
        x_train_sequence = tokenizer.texts_to_sequences(x_train)
        x_test_sequence = tokenizer.texts_to_sequences(x_test)

        train_padded = pad_sequences(x_train_sequence, maxlen = 600,
                                     dtype = 'int32', truncating = 'post',
                                     padding = 'post')
        test_padded = pad_sequences(x_test_sequence, maxlen = 600,
                                    dtype = 'int32', truncating = 'post',
                                    padding = 'post')

    except:
        print('Error al convertir los textos en secuencias')
        raise

    print('Secuenciación y padding realizados con éxito')

    try:
        print('\n--- Carga de TensorBoard y limpieza de logs previos')
        # Cargamos la extensión de TB para notebooks
        %load_ext tensorboard

        # Limpiar logs de ejecuciones anteriores
        !rm -rf ./logs/
    except:
        print('Error al cargar TensorBoard')
        raise
    print('TensorBoard Cargado correctamente')

    try:
        print('\n--- Definición de los hiperparámetros a optimizar')
        HP_DROPOUT = hp.HParam('dropout', hp.RealInterval(0.2, 0.4))
        METRIC_F1 = 'f1-score'

        with tf.summary.create_file_writer('logs/hparam_tuning').as_default():
            hp.hparams_config(
            hparams = [HP_DROPOUT],
            metrics = [hp.Metric(METRIC_F1, display_name = 'F1-Score')])
    except:
        print('Error al definir los hiperparámetros')
        raise
    print('Hiperparámetros definidos correctamente')

    try:
        print('\n--- Ejecución Grid Search')
        session_num = 0

        for dropout_rate in (HP_DROPOUT.domain.min_value, HP_DROPOUT.domain.max_value):
            hparams = {'HP_DROPOUT': dropout_rate}

            run_name = 'run-%d' % session_num
            print('--Iniciando ejecución : %s' % run_name)
            print({h: hparams[h] for h in hparams})
            run('logs/hparam_tuning/' + run_name,
                hparams, train_padded, test_padded, y_train, y_test)
            session_num += 1
    except:
        print('Error al realizar Grid Search')
        raise

    print('\n----------------------------------------------------------------------')
    print('Grid Search realizado con éxito')
```

Ilustración 160: Código Prueba 11 - Parte 2

En la Ilustración 161 se muestran los resultados de la prueba.

```
In [6]: prueba11()

        Prueba 11: Grid Search con un subset de los hiperparámetros

        --- Lectura del dataset preprocesado
        Dataset Leído correctamente

        --- División de los datos en entrenamiento y pruebas
        Datos separados correctamente

        --- Creación del diccionario (Vocabulario)
        Vocabulario creado correctamente

        --- Transformar los textos en secuencias y padding
        Secuenciación y padding realizados con éxito

        --- Carga de TensorBoard y limpieza de logs previos
        TensorBoard Cargado correctamente

        --- Definición de los hiperparámetros a optimizar
        Hiperparámetros definidos correctamente

        --- Ejecución Grid Search
        --Iniciando ejecución : run-0
        {'HP_DROPOUT': 0.2}
        Train on 13596 samples
        13596/13596 [==============================] - 564s 41ms/sample - loss: 0.3003 - accuracy: 0.8901 - Precision: 0.8824 - Recall:
        0.9382 - f1_score: 0.9109
        5827/5827 [==============================] - 70s 12ms/sample - loss: 0.1333 - accuracy: 0.9475 - Precision: 0.9530 - Recall: 0.
        9558 - f1_score: 0.9529
        --Iniciando ejecución : run-1
        {'HP_DROPOUT': 0.4}
        Train on 13596 samples
        13596/13596 [==============================] - 564s 41ms/sample - loss: 0.3292 - accuracy: 0.8800 - Precision: 0.8639 - Recall:
        0.9449 - f1_score: 0.9072
        5827/5827 [==============================] - 70s 12ms/sample - loss: 0.1332 - accuracy: 0.9490 - Precision: 0.9575 - Recall: 0.
        9537 - f1_score: 0.9540

        ----------------------------------------------------------------------
        Grid Search realizado con éxito
```

Ilustración 161: Resultados de la Prueba 11

9.5. Resumen

La consecución de este último Sprint ha llevado un total de 63 horas, más otras 7 de investigación. Durante este Sprint se ha llevado a cabo la optimización de hiperparámetros de la Red Neuronal Implementada a lo largo del Sprint 3 y se ha reestructurado el código de los notebooks para que sea ejecutable mediante scripts de Python.

Además de esto, también se han redactado los manuales de usuario e instalación, así como las conclusiones del proyecto y las desviaciones temporales y de costes con respecto a la planificación inicial.

Como puede observarse en el gráfico Burndown de la Ilustración 162, el ritmo de trabajo en este sprint ha estado muy alejado del ideal, esto se debe principalmente a la gran cantidad de tiempo de ejecución que requiere la optimización de hiperparámetros, además de los errores que ha generado, lo que ha hecho que haya sido necesario reestructurar el código y comenzar de nuevo la ejecución en varias ocasiones.

A pesar de este contratiempo, todas las tareas asignadas para este sprint han sido cumplidas dentro de los plazos establecidos.

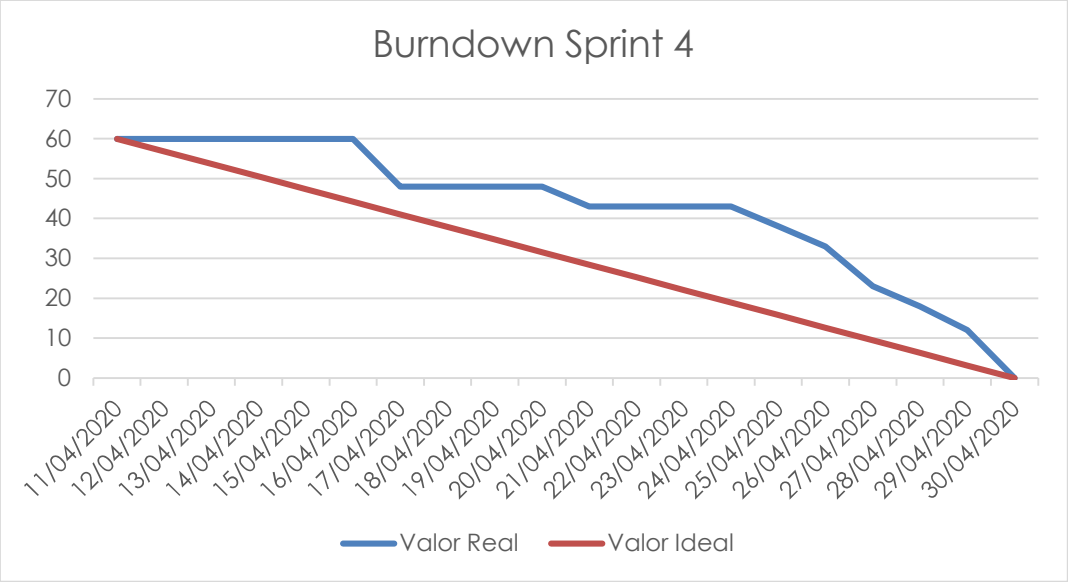


Ilustración 162: Burndown Sprint 4

PARTE IV. MANUALES

10. Manual de instalación

En esta sección se detallarán los pasos necesarios para instalar todos los componentes requeridos para llevar a cabo la ejecución del proyecto desarrollado.

10.1. Requisitos hardware

Trabajar con algoritmos de Machine Learning y Deep Learning exige tener unos componentes hardware medianamente potentes. Los requisitos mínimos recomendados son los siguientes:

- **Procesador (CPU):** Intel Core i5 de sexta generación o superior.
- **Memoria RAM:** Se recomienda tener 16 GB o más para la ejecución de los modelos más potentes, para el análisis del dataset y la mayoría de las pruebas es suficiente con 8 GB.
- **Sistema Operativo:** Ubuntu, MAC o Windows 10. Se recomienda tener instaladas las últimas actualizaciones del sistema operativo que se esté usando. Para el desarrollo de este proyecto se ha empleado Windows 10.

10.1.1. Hardware usado

Como referencia, a continuación se muestran los componentes hardware de los equipos empleados para el desarrollo.

Componentes Equipo 1 (Sobremesa)	Componentes Equipo 2 (Portátil)
Procesador Intel Core i7-6700 a 3.4GHz	Procesador Intel Core i5-8250U a 1.6GHz
16GB de Memoria RAM	8GB de Memoria RAM
Sistema Operativo Windows 10 Pro	Sistema Operativo Windows 10 Pro
Disco Duro SSD 120GB	Disco Duro 1TB

Tabla 63: Componentes Hardware de los equipos

Para las pruebas de los Sprints 1 a 3 y para el análisis del dataset, el Equipo 2 ha sido más que suficiente. No obstante, ha sido insuficiente para realizar la ejecución de los modelos neuronales más avanzados.

10.2. Requisitos software

Las librerías necesarias para ejecutar el código de este proyecto son las siguientes. Todas estas librerías vienen incluidas en el fichero de entorno virtual de Anaconda del repositorio (*entorno-tfg.yml*)

- Python 3.7.3
- Gensim 3.8.1
- Keras 2.3.1
- Nltk 3.4.4
- Numpy 1.18.1
- Tensorflow 2.1.0
- Tensorboard 2.1.1
- Pandas 0.24.2
- Matplotlib 3.1.0
- Seaborn 0.9.0
- Scikit-learn 0.21.2

10.3. Proceso de instalación

10.3.1. Instalación de Anaconda

Para instalar Anaconda, nos dirigimos a https://www.anaconda.com/products/individual y elegimos la versión de Python 3.7 que se adecue al sistema operativo y arquitectura usados. En este caso, Windows 10 de 64 bits.

Ilustración 163: Descarga del Instalador de Anaconda

Una vez finalizada la descarga, se ejecuta el instalador *"Anaconda3-2020.02-Windows-x86_64.exe"* (el nombre puede variar ligeramente en función de la versión descargada).

Ilustración 164: Instalador de Anaconda

El proceso de instalación es bastante sencillo, simplemente tenemos que pulsar el botón Next y aceptar los términos y condiciones, deben dejarse los valores por defecto de la configuración. Si la instalación se completa con éxito, aparecerá la ventana mostrada en la Ilustración 165.

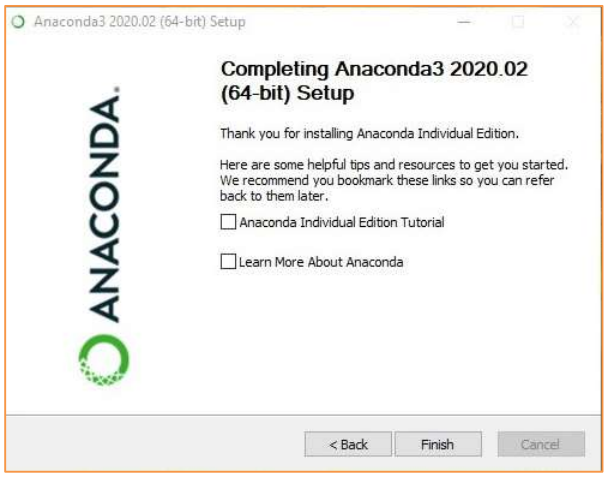


Ilustración 165: Instalación de Anaconda completada con éxito

10.3.2. Importación y activación del entorno virtual

En el repositorio del proyecto se ha facilitado el entorno virtual que contiene todas las librerías que se han estado usando, el nombre del archivo es "*entorno-tfg.yml*".

Para importar el entorno virtual, abrimos Anaconda Navigator y, en el menú de la izquierda, se accede a "*Environments*"(Ilustración 166):

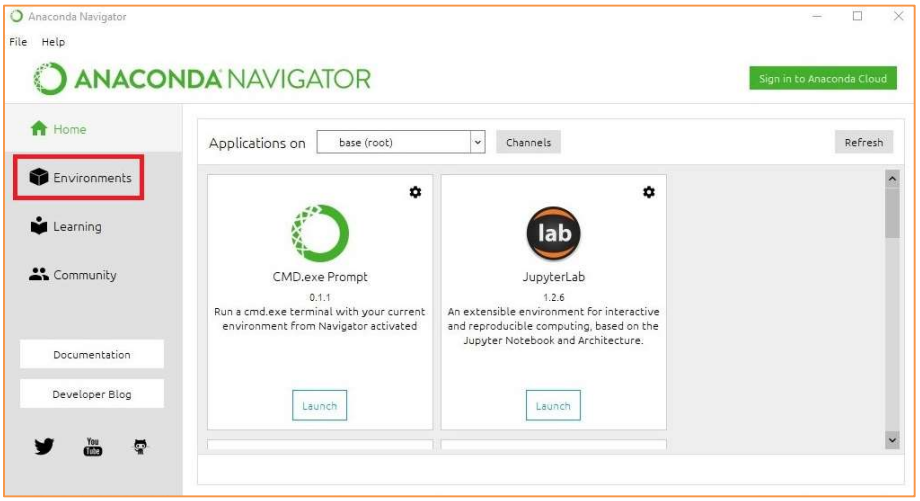


Ilustración 166: Pantalla principal de Anaconda Navigator

En la pestaña "Environments", aparecerá activo por defecto el entorno base (root) y, justo en la parte inferior aparecen cuatro botones: "*Create*", "*Clone*", "*Import*" y "*Remove*". Pulsamos sobre el botón "*Import*"

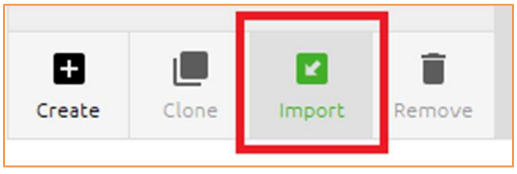


Ilustración 167: Botón Import

Tras pulsar el botón, aparecerá una ventana en la que se nos indicará que insertemos la ruta del entorno y el nombre que va a recibir.

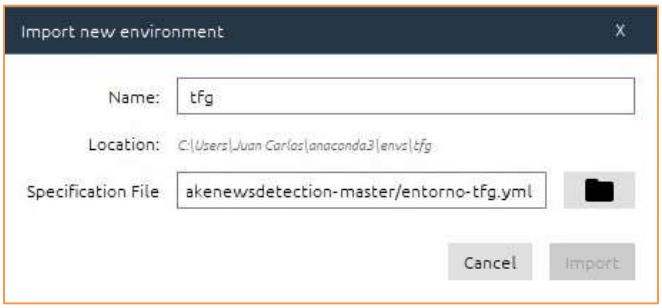


Ilustración 168: Importación del entorno

Para activar el entorno abrimos Anaconda Prompt en el menú de Windows. Para activar un entorno escribimos "*conda activate environment_name*" reemplazando environment_name por el nombre del entorno (Ilustración 169).

Para comprobar el nombre del entorno podemos escribir "*conda info --env*" en Anaconda Prompt.

Ilustración 169: Entorno de ejecución activado

Por último, debemos instalar algunos recursos adicionales de la librería nltk que no se encuentran en el entorno de ejecución y son necesarios para la ejecución del preprocesado del conjunto de datos. Para ello, ejecutamos los siguientes comandos dentro de Anaconda Prompt teniendo el activo el entorno de ejecución del proyecto.

python

>>>import nlt

>>>nltk.download('popular')

>>>exit()

exit

La descarga e instalación de los recursos se muestra en las Ilustraciones 170 y 171.

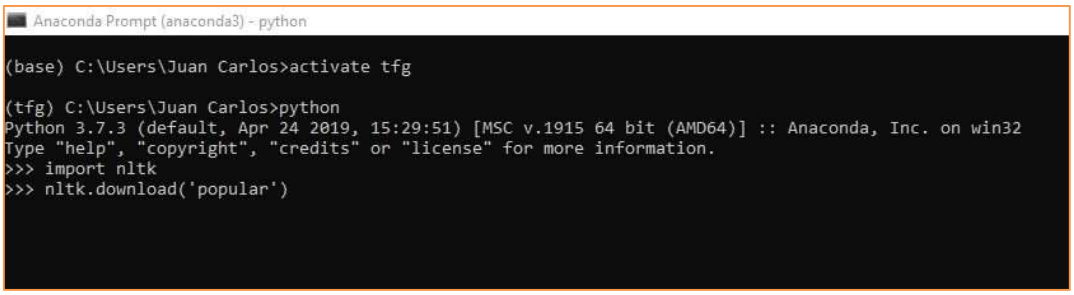


Ilustración 170: Instalación de recursos nltk (Parte 1)

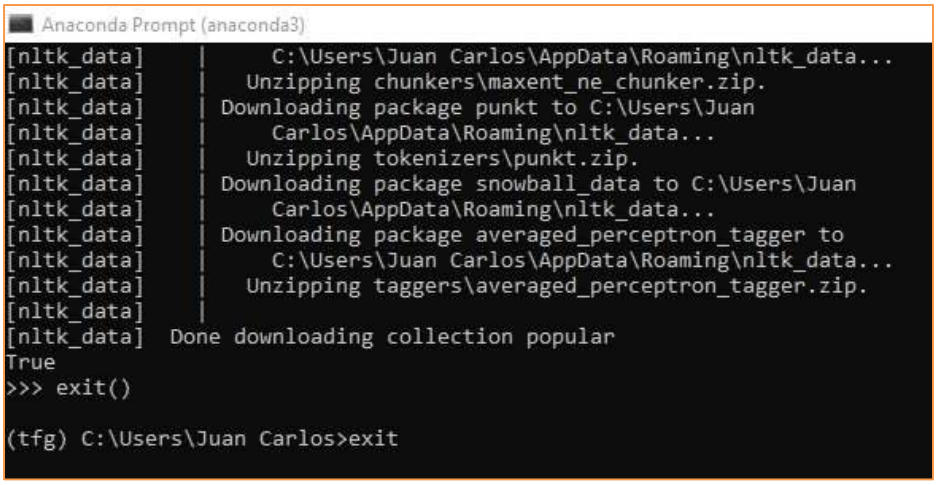


Ilustración 171: Instalación de recursos nltk (Parte 2)

11. Manual de usuario

11.1. Ejecución en Jupyter Notebook

Los Jupyter Notebooks se encuentran dentro de las carpetas "analisis y preprocesado", "notebooks" y "pruebas" del repositorio, siguiendo la siguiente estructura de directorios:

- **analisis y preprocesado:**
  - *Analisis TI-CNN dataset (Parte 1).ipynb*
  - *Analisis TI-CNN dataset (Parte 2).ipynb*
  - *Preprocesado.ipynb*
- **notebooks:**
  - **BagOfWords:**
    - *Binary Count ticnn functional.ipynb*
    - *Term Frequency Functional ticnn.ipynb*
    - *TF-IDF ticnn functional.ipynb*
  - **Doc2Vec:**
    - *Doc2Vec (DBOW).ipynb*
    - *Doc2Vec (Distributed Memory).ipynb*
  - **Naive Bayes:**
    - *Naive Bayes ticnn functional.ipynb*
  - **RecurrentNeuralNetwork:**
    - *Análisis de los logs.ipynb*
    - *GridSearch.ipynb*
    - *Red Neuronal Recurrente.ipynb*
  - **Word2Vec:**
    - *Word2Vec (CBOW) ticnn functional.ipynb*
    - *Word2Vec (Skipgram) ticnn functional.ypynb*
- **pruebas:**
  - *Pruebas Sprint 1.ipynb*
  - *Pruebas Sprint 2.ipynb*
  - *Pruebas Sprint 3.ipynb*
  - *Pruebas Sprint 4.ipynb*

Para la realización del proyecto, se ha utilizado Jupyter notebook para el análisis y preprocesado de los datos; las pruebas y la optimización de hiperparámetros, empleando Google Colab para los modelos neuronales. No obstante, los notebooks con los modelos neuronales han sido adaptados para que se puedan ejecutar con Jupyter Notebook.

Deben cumplirse las siguientes condiciones para la correcta ejecución de estos notebooks:

- El fichero "*nlp_functions.py*" debe encontrarse en el directorio raíz del proyecto.
- Los archivos "*all_data.csv*" y "*ticnn_preprocessed.csv*" deben encontrarse dentro de la carpeta "dataset", ubicada en el directorio raíz del proyecto.
- El fichero "*subset_test.csv*" debe encontrarse dentro de la carpeta "pruebas".

La distribución de los directorios en el repositorio cumple todas las condiciones enumeradas anteriormente, exceptuando el fichero "*all_data.csv*", que al superar los 100 MB no ha podido ser subido a GitHub, este archivo puede encontrarse en [53].

11.1.1. Inicialización de Jupyter Notebook

Una vez activado el entorno de ejecución de Python, para acceder a Jupyter Notebook simplemente tenemos que abrir Anaconda Prompt, escribir "*jupyter-notebook*" y presionar *Enter* (Ilustración 172).

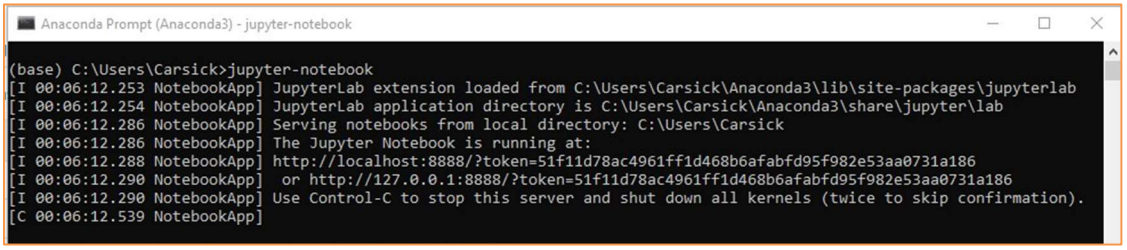


Ilustración 172: Inicialización de Jupyter Notebook

11.1.2. Abrir los notebooks

En la pestaña que se habrá abierto en el navegador, nos desplazamos a través de los directorios hasta llegar a la carpeta en la que se encuentre el notebook que queremos ejecutar. Haciendo clic en él, se abrirá en una nueva pestaña (Ilustración 173)

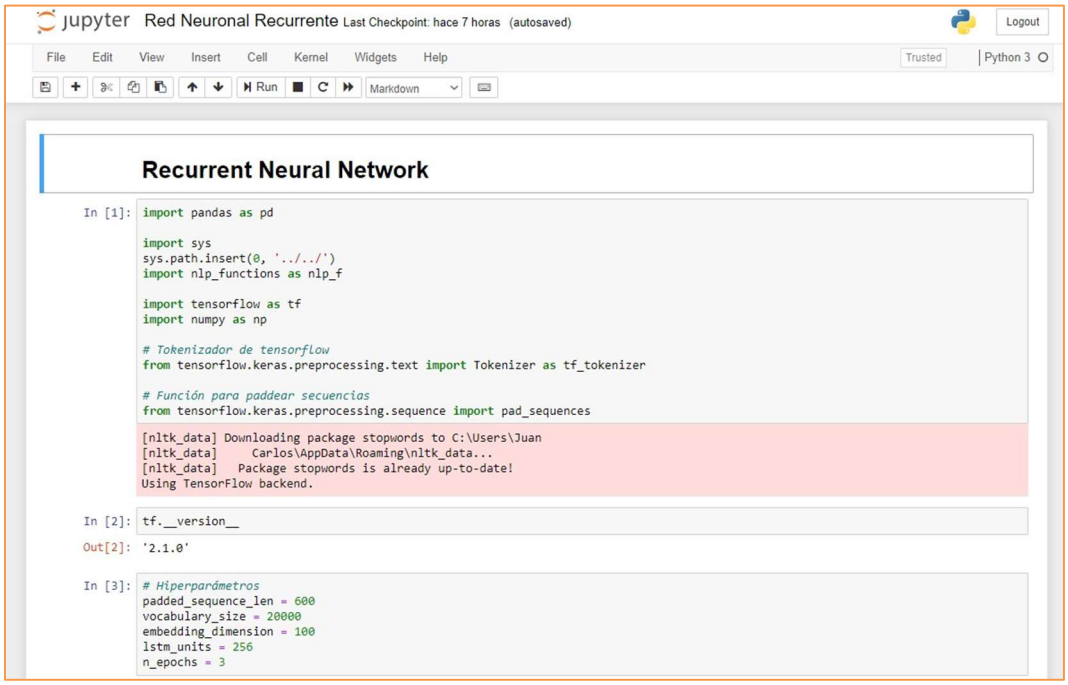


Ilustración 173: Notebook de Jupyter

Para ejecutar una celda, la seleccionamos y pulsamos "Ctrl+Enter", o bien pulsamos en el botón "Run" que aparece en el menú superior. El orden de ejecución es lineal (de arriba abajo).

Para ejecutar el notebook completo, pulsamos "Cell → Run all" (Ilustración 174)

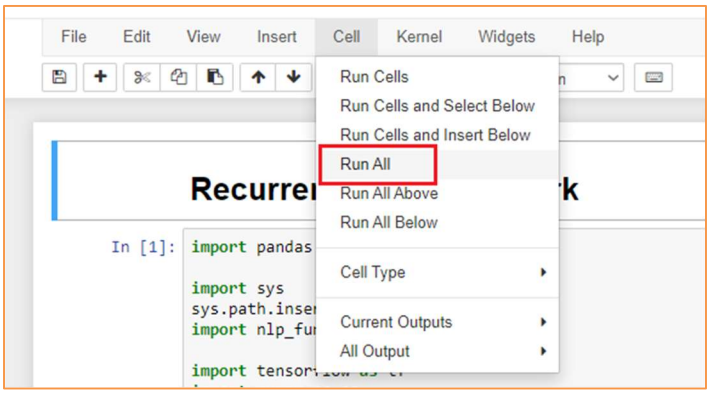


Ilustración 174: Ejecutar todas las celdas de un notebook

11.2. Ejecución en Google Colab

Aunque los notebooks de los modelos neuronales han sido modificados para poder ser ejecutados en Scripts de Python y en Notebooks de Jupyter, originalmente fueron implementados en Google Colaboratory.

Estos notebooks tienen la ventaja de que no es necesario tener instalados los componentes indicados en la sección Requisitos software para ser ejecutados. Cabe destacar que estos notebooks son solo los modelos neuronales, ya que el análisis del dataset y las pruebas se han implementado directamente en Jupyter, al requerir de menos recursos computacionales.

El primer paso para ejecutar estos notebooks es iniciar sesión en Google Drive (https://www.google.com/intl/es_ALL/drive/) con las credenciales de la cuenta de Google indicada en la Tabla 64.

Credenciales Cuenta de Google	
Nombre de usuario	Juaaloval.tfg@gmail.com
Contraseña	68414e77c90db9d24a404327e6cecc6c

Tabla 64: Credenciales Cuenta de Google

Tras acceder, nos encontraremos ante un sistema de directorios que seguirá la distribución que se muestra a continuación:

- **Modelos Neuronales**
  - **Bag Of Words**
    - *Binary Count ticnn functional.ipynb*
    - *Term Frequency Functional ticnn.ipynb*
    - *TF-IDF ticnn functional.ipynb*
  - **Dataset**
    - *Ticnn_preprocessed.csv*
  - **Doc2Vec**
    - *Doc2Vec (DBOW).ipynb*
    - *Doc2Vec (Distributed Memory).ipynb*
  - **Naive Bayes**

        - ➢ *Naive Bayes ticnn functional.ipynb*
    - **Recurrent Neural Network**
        - ➢ *Red Neuronal Recurrent 10 epochs.ipynb*
    - **Word2Vec**
        - ➢ *Word2Vec (CBOW) ticnn functional.ipynb*
        - ➢ *Word2Vec (Skipgram) ticnn functional.ipynb*
- *nlp_functions.py*

Para abrir un notebook situamos el ratón sobre él y pulsamos botón derecho → Abrir con → Google Colaboratory.

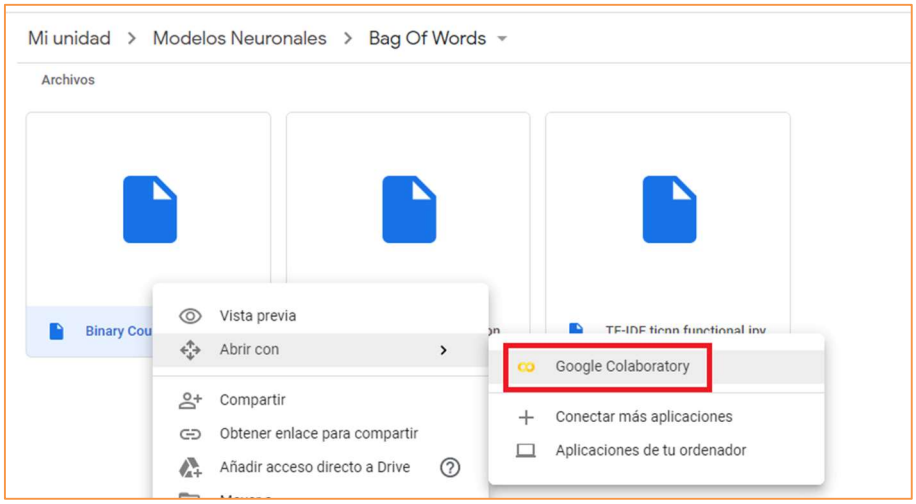


Ilustración 175: Abrir un notebook con Google Colab

Antes de ejecutar el notebook, es necesario conectarse a un entorno de ejecución, para ello pulsamos sobre el botón "Conectar" de la esquina superior derecha de la pantalla (Ilustración 176). Tras unos segundos, se nos informará de que estamos conectados a un entorno de ejecución de 12GB de RAM (Ilustración 177).

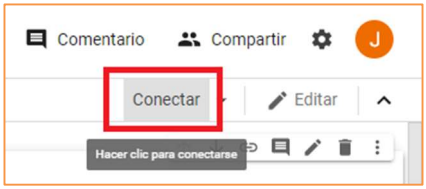


Ilustración 176: Conectar a Entorno de ejecución

Ilustración 177: Conexión realizada con éxito

Para poder acceder a archivos externos al notebook actual (como es el caso del dataset y el archivo nlp_functions.py), pulsamos en el icono de la carpeta situado en el lateral izquierdo de la pantalla y seleccionamos "Activar Drive" (Ilustración 178).

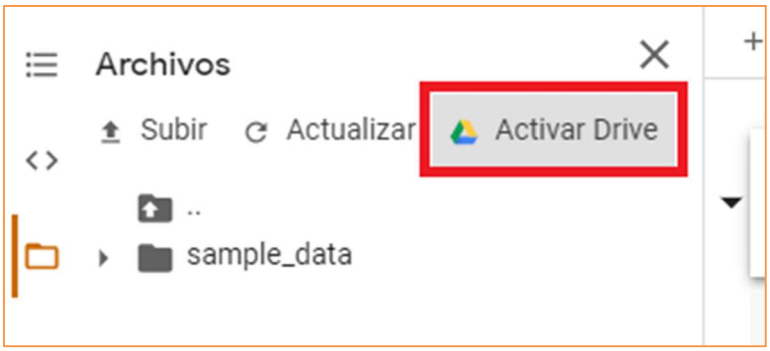


Ilustración 178: Activación de Google Drive

En determinadas ocasiones puede ocurrir que se genere una celda al principio del notebook para que la ejecutemos. Lo hacemos pulsando sobre el botón de ejecutar, tal y como se nos indica en la Ilustración 179. Nos aparecerá una url y un cuadro de texto.

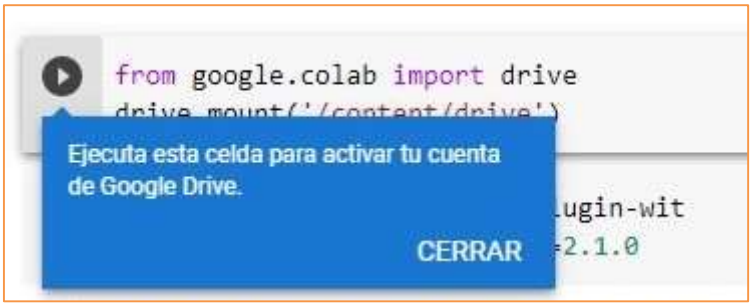


Ilustración 179: Código para activar Google Drive

Si pulsamos sobre la url se nos llevará a una página en la que nos pedirán aceptar unos términos y condiciones, pulsamos en permitir y aparecerá un código que deberemos pegar en el cuadro de texto mencionado anteriormente (Ilustración 180).

Ilustración 180: Código único

Pulsamos Enter y deberá aparecer un mensaje informándonos de que Drive se ha activado correctamente (Ilustración 181).

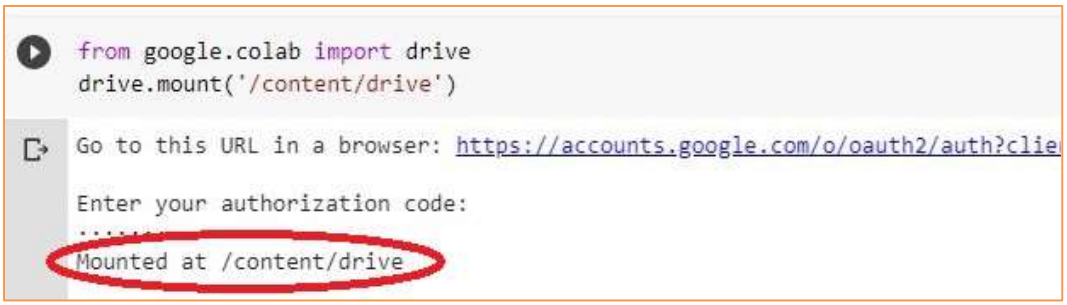


Ilustración 181: Drive activado con éxito

Para confirmar que Drive se ha activado correctamente, pulsamos de nuevo sobre el icono de la carpeta de la izquierda. Además de las carpetas anteriores, deberá aparecer una nueva carpeta llamada “drive” (Si hemos dejado abierta esta pestaña es posible que no aparezca, pero podrá verse si la cerramos y la volvemos a abrir).

Ilustración 182: Carpeta Drive

Al igual que en Jupyter Notebook, para ejecutar una celda, se selecciona el código que contiene y se presiona Ctrl+Enter. De nuevo, el orden de ejecución es lineal (de arriba a abajo).

Pueden ejecutarse todas las celdas en orden lineal desde el menú Entorno de Ejecución → Reiniciar y Ejecutar Todo (Ilustración 183)

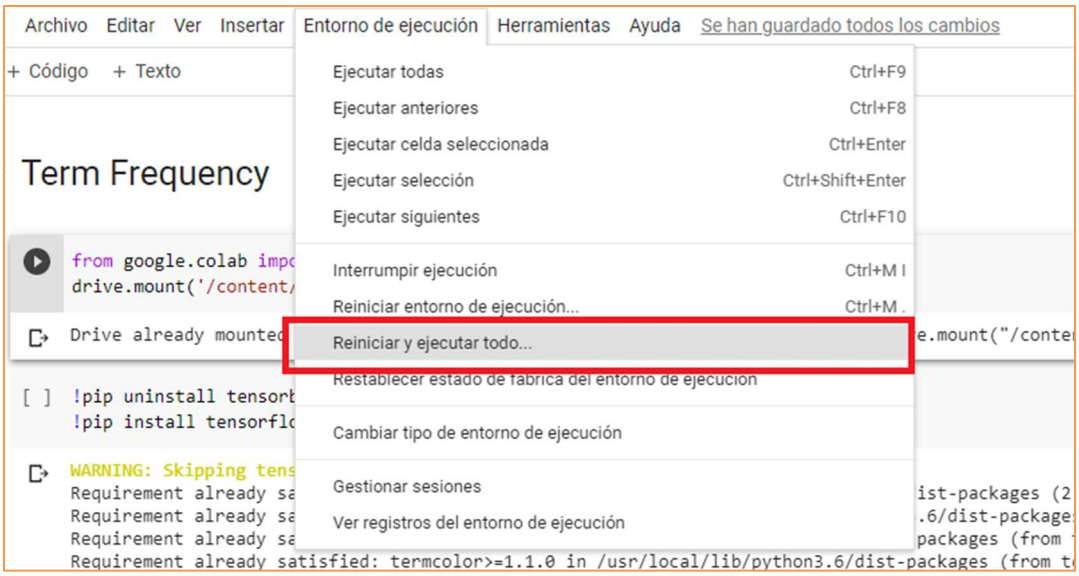


Ilustración 183: Ejecutar Notebook Completo

Al ejecutar el notebook, puede ocurrir que TensorBoard no se inicie correctamente, al tener almacenados logs de una ejecución anterior. Para solucionar esto TensorBoard nos indica que ejecutemos el comando "!kill xyz" en una celda separada para eliminar el proceso anterior (Donde xyz es el puerto en el que se está ejecutando TensorBoard). Una vez ejecutado dicho comando, podemos volver a ejecutar TensorBoard.

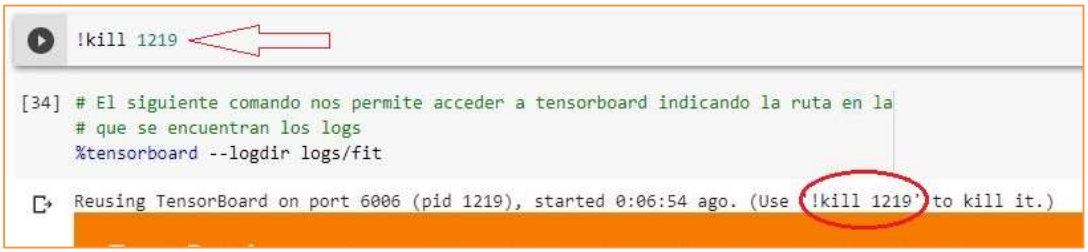
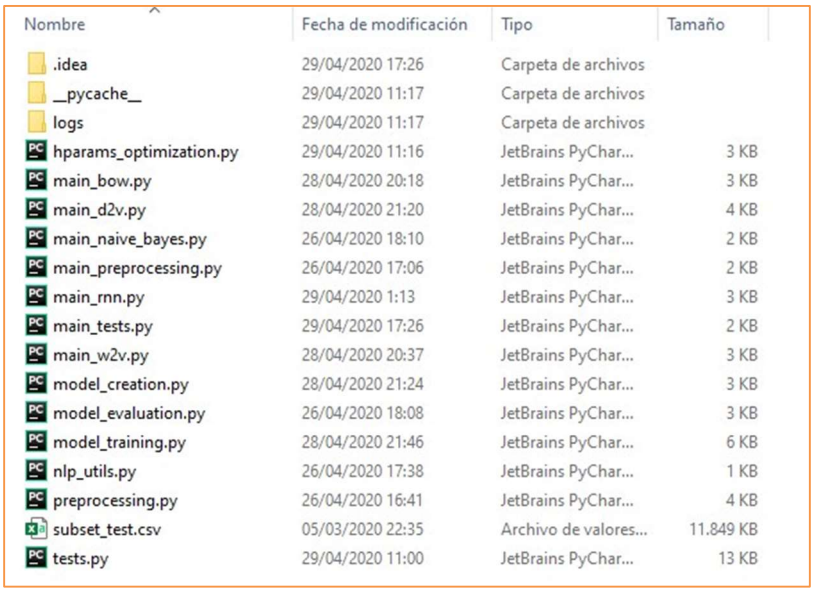


Ilustración 184: Reactivación de TensorBoard

11.3. Ejecución de los scripts de Python

Los scripts de Python se encuentran en la carpeta "python_scripts" y contienen las funciones indicadas en la sección Estructuración del código desarrollado del Sprint 4.



Nombre	Fecha de modificación	Tipo	Tamaño
.idea	29/04/2020 17:26	Carpeta de archivos	
__pycache__	29/04/2020 11:17	Carpeta de archivos	
logs	29/04/2020 11:17	Carpeta de archivos	
hparams_optimization.py	29/04/2020 11:16	JetBrains PyChar...	3 KB
main_bow.py	28/04/2020 20:18	JetBrains PyChar...	3 KB
main_d2v.py	28/04/2020 21:20	JetBrains PyChar...	4 KB
main_naive_bayes.py	26/04/2020 18:10	JetBrains PyChar...	2 KB
main_preprocessing.py	26/04/2020 17:06	JetBrains PyChar...	2 KB
main_rnn.py	29/04/2020 1:13	JetBrains PyChar...	3 KB
main_tests.py	29/04/2020 17:26	JetBrains PyChar...	2 KB
main_w2v.py	28/04/2020 20:37	JetBrains PyChar...	3 KB
model_creation.py	28/04/2020 21:24	JetBrains PyChar...	3 KB
model_evaluation.py	26/04/2020 18:08	JetBrains PyChar...	3 KB
model_training.py	28/04/2020 21:46	JetBrains PyChar...	6 KB
nlp_utils.py	26/04/2020 17:38	JetBrains PyChar...	1 KB
preprocessing.py	26/04/2020 16:41	JetBrains PyChar...	4 KB
subset_test.csv	05/03/2020 22:35	Archivo de valores...	11.849 KB
tests.py	29/04/2020 11:00	JetBrains PyChar...	13 KB

Ilustración 185: Scripts de Python

Para la correcta ejecución de estos scripts, deben cumplirse las siguientes condiciones:

- El fichero "*subset_test.csv*" debe encontrarse en la misma carpeta que los scripts.
- Los archivos "*all_data.csv*" y "*ticnn_preprocessed.csv*" deben estar dentro de la carpeta "dataset", ubicada en el directorio raíz del proyecto.

De nuevo, la distribución inicial de los directorios del repositorio cumple con estas condiciones, exceptuando el fichero "*all_data.csv*", que puede encontrarse en [53].

Para ejecutar cada uno de estos modelos (así como el preprocesado del dataset y las pruebas), debemos ejecutar su correspondiente fichero main, que se encarga de llamar a las funciones necesarias.

11.4. Manual de uso de TensorBoard

11.4.1. Iniciar TensorBoard

11.4.1.1. Scripts de Python

Una vez finalizado el entrenamiento y la evaluación de un modelo mediante los scripts de Python, se generan unos logs con los resultados dentro de la carpeta "logs" de "python_scripts" (en el repositorio original ya hay logs de cada uno de los modelos). Cada ejecución se ubica dentro de la carpeta correspondiente al tipo de modelo entrenado (por ejemplo, para la red neuronal recurrente los logs se almacenarán dentro de la subcarpeta "rnn").

Dentro de cada carpeta, los logs de distintas ejecuciones se diferenciarán entre sí almacenándose en directorios diferentes según la fecha y la hora de creación, siguiendo el formato: "año,mes,día-hora,minuto,segundo". Dentro de cada una de estas carpetas, encontramos los directorios train y validation.

Para iniciar TensorBoard, abrimos Anaconda Prompt y escribimos el comando “tensorboard –logdir my_dir”, siendo my_dir la carpeta en la que están almacenados los logs. La Ilustración 186 muestra como iniciar TensorBoard utilizando la ruta de los logs de la Red Neuronal Recurrente (es importante tener en cuenta que la inicialización de TensorBoard nos dará error si uno de los directorios de la ruta contiene espacios):

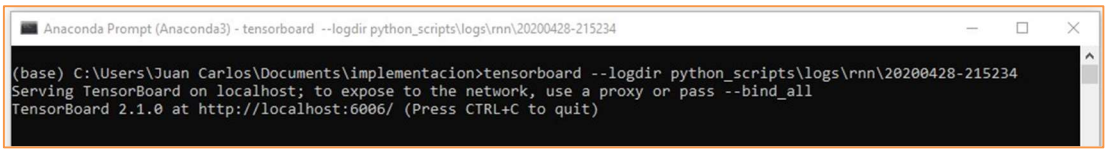


Ilustración 186: Inicialización de TensorBoard en Anaconda Prompt

Pulsamos Enter y se nos indicará que TensorBoard se ha iniciado correctamente en un puerto, que por defecto es el 6006, escribiendo localhost:6006 en el navegador, accedemos al panel de TensorBoard.

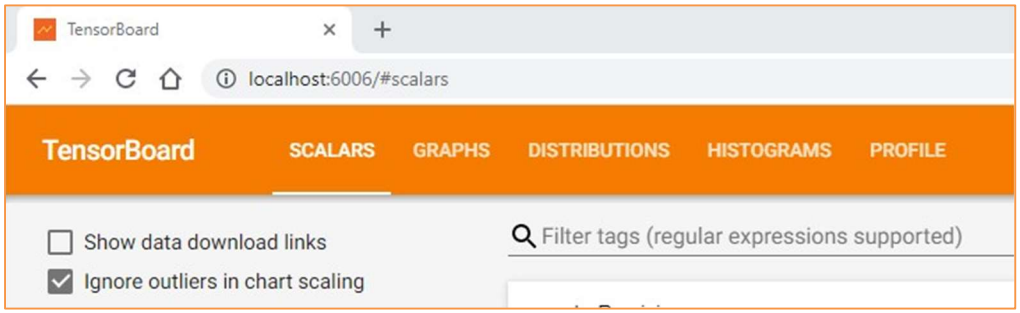


Ilustración 187: TensorBoard en el navegador

Para cerrar TensorBoard, volvemos a Anaconda Prompt y pulsamos “Ctrl+C”.

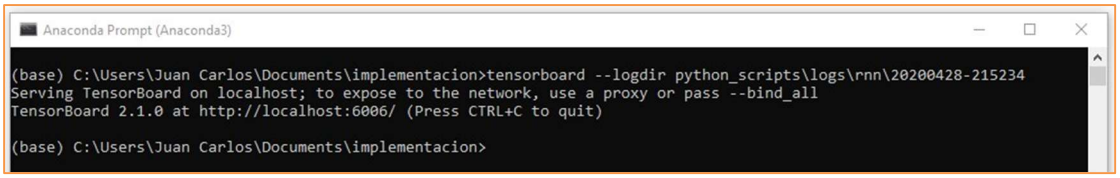


Ilustración 188: Cerrar TensorBoard

11.4.1.2. Jupyter Notebooks

Si pulsamos Run All dentro del notebook de Jupyter para ejecutar todas las celdas, TensorBoard se iniciará al final de notebook, es bastante común que dé un error por timeout, en cuyo caso volvemos a ejecutar la celda pulsando “Ctrl+Enter” para que se muestre.

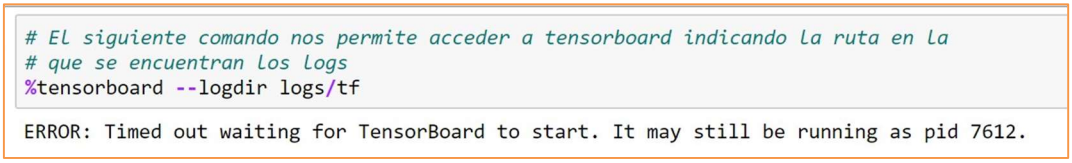


Ilustración 189: Timeout de TensorBoard en Jupyter Notebook

Además de visualizar TensorBoard en el notebook, podemos abrir una nueva pestaña del navegador y escribir “localhost:6006” (o el puerto que se nos indique) para visualizarlo a pantalla completa.

Es importante tener en cuenta que, a diferencia de en los scripts de Python, en los notebooks de Jupyter solo se almacenan los logs de la última ejecución para evitar conflictos. En el caso de que queramos visualizar los resultados de un segundo notebook, debemos cerrar el kernel del primero, o TensorBoard no se activará correctamente.

Para cerrar el kernel de un notebook, lo seleccionamos en el menú de Jupyter Notebook (el icono del notebook deberá estar en verde) y pulsamos en “shutdown”:

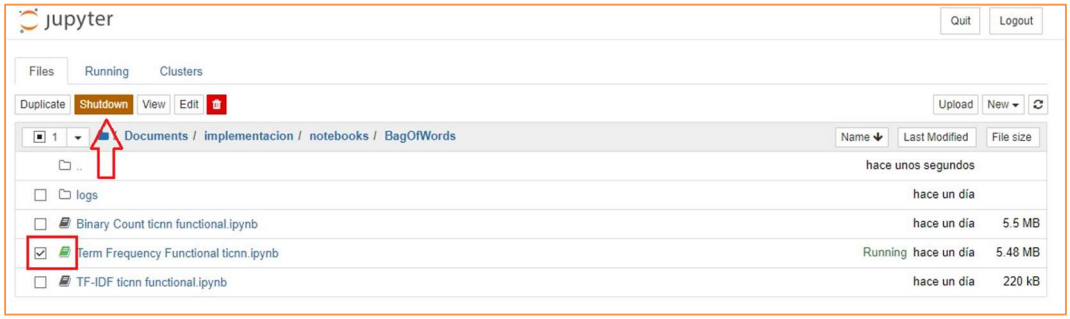


Ilustración 190: Cerrar un kernel de Jupyter Notebook

También podemos acceder a TensorBoard siguiendo los mismos pasos que en el apartado Scripts de Python si escribimos la ruta de los logs generados por el cuaderno de Jupyter en Anaconda Prompt.

11.4.2. Interfaz de TensorBoard

11.4.2.1. Análisis de resultados

Una vez iniciado TensorBoard, nos encontramos con una interfaz como la de la Ilustración 191, con cuatro pestañas: scalars, graphs, distributions e histograms. Para los análisis realizados durante este proyecto, se han empleado principalmente scalars y graphs, pero se realizará una breve descripción de distributions e histograms. Puede ocurrir que aparezca una quinta pestaña llamada profiles, se trata de una pestaña opcional que no ha sido utilizada en este proyecto, pero a veces se muestra la opción de seleccionarla, aunque estará en blanco.

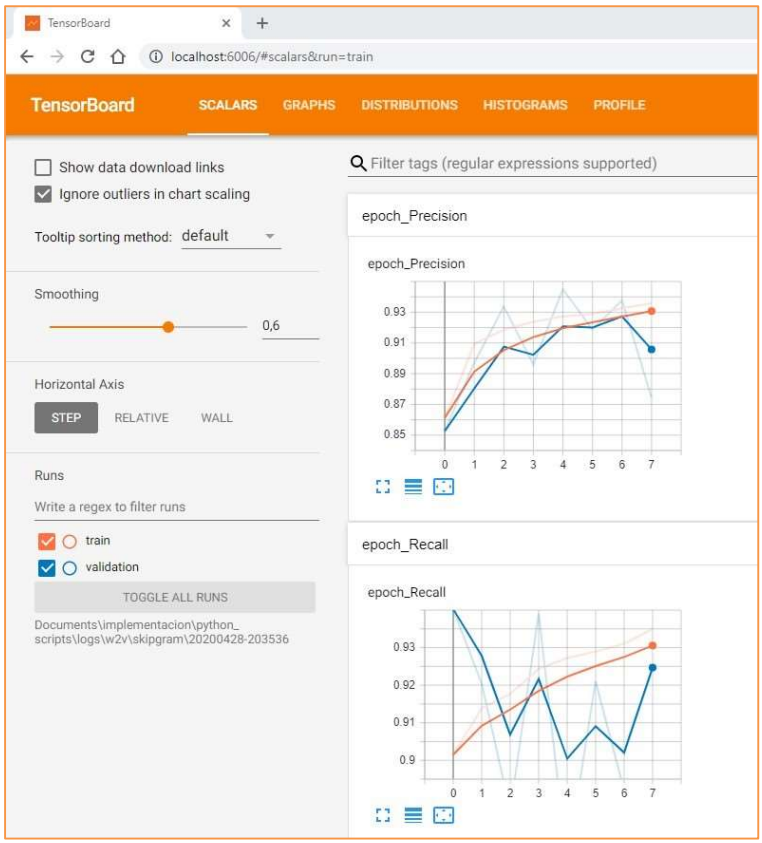


Ilustración 191: Pestaña Scalars de TensorBoard

- **Scalars (Ilustración 191):** nos permite analizar la evolución de las distintas métricas (accuracy, precision, recall y F1-Score) a lo largo de todas las épocas tanto en el conjunto de entrenamiento como en el de pruebas. En la pestaña de la izquierda hay dos checkbox con los que se pueden y mostrar las gráficas en ambos conjuntos. También puede usarse la barra de Smoothing para alterar el suavizado de las gráficas.
- **Graphs (Ilustración 192):** Contiene una representación gráfica del grafo de computación y de las funciones empleadas. El panel de la izquierda, además de mostrar la leyenda, nos permite realizar acciones simples como cambiar el color del grafo y descargarlo como png.

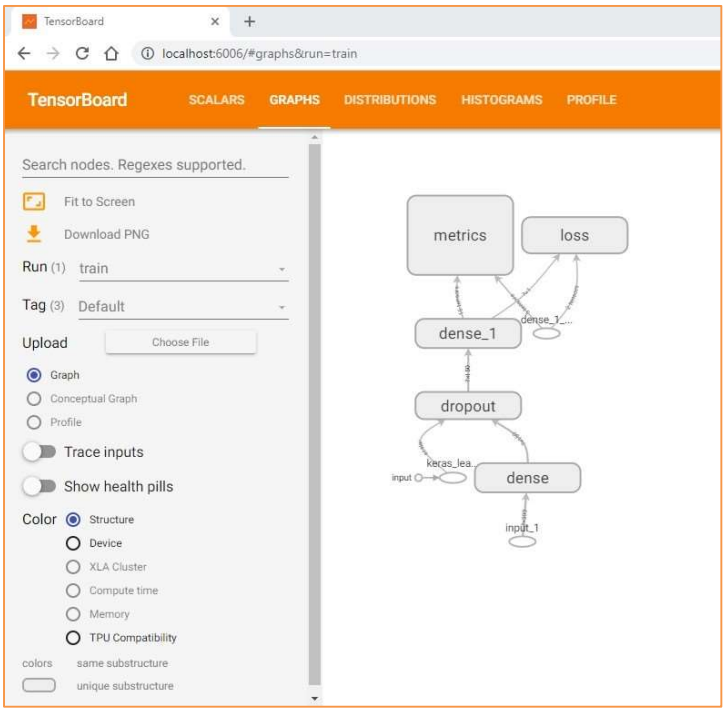


Ilustración 192: Pestaña Graphs de TensorBoard

- **Distributions (Ilustración 193) e Histogram (Ilustración 194):** Muestran la distribución de los tensores a lo largo del tiempo. Esto puede resultar útil para visualizar y verificar que los pesos y los sesgos de las distintas capas evolucionan como se espera.

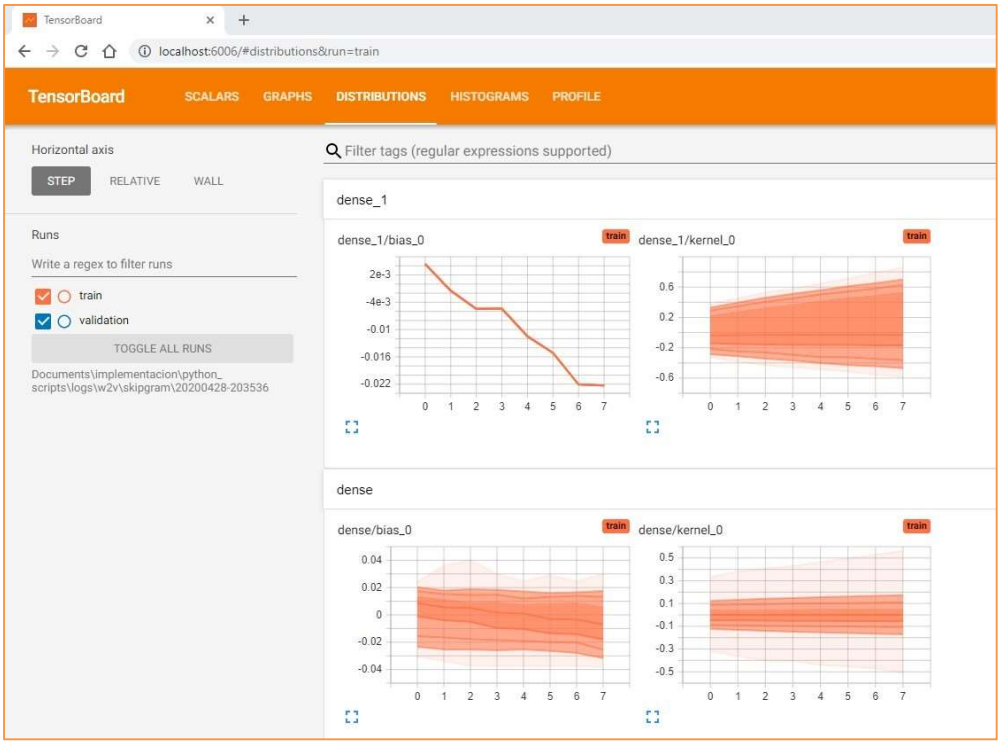


Ilustración 193: Pestaña Distributions de TensorBoard

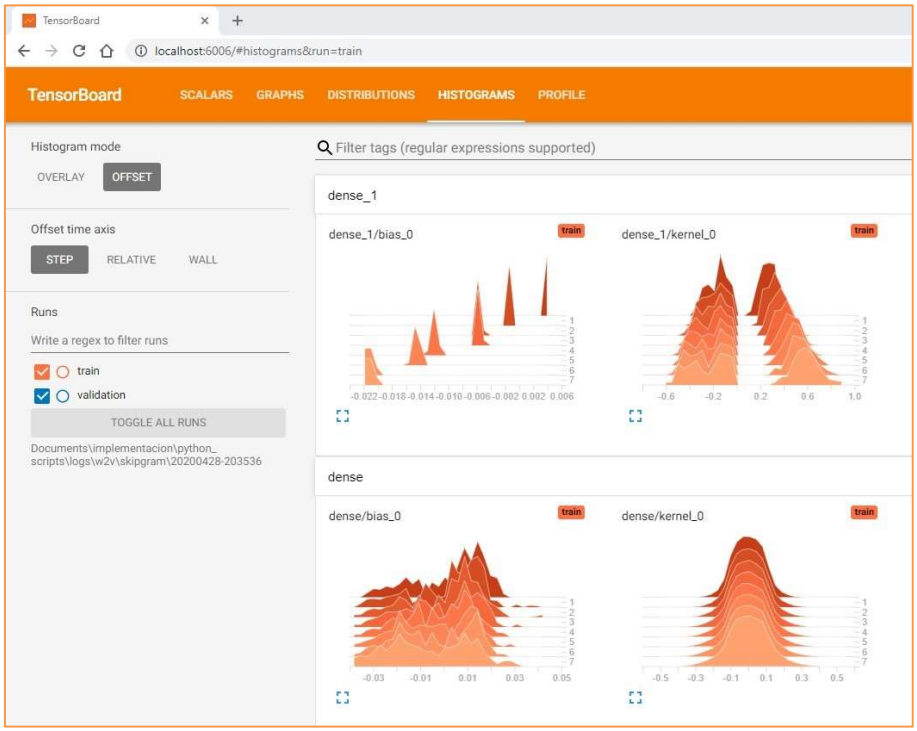


Ilustración 194: Pestaña Histograms de TensorBoard

11.4.2.2. Optimización de hiperparámetros

En lo que respecta a la visualización en TensorBoard de los resultados de la Optimización de Hiperparámetros (los logs se encuentran en la carpeta "notebooks\RecurrentNeuralNetwork\logs\hparam_tuning"), el panel principal tiene dos pestañas:

- **Scalars (Ilustración 195):** Idéntico al panel del mismo nombre explicado en el apartado anterior. En este caso solo nos muestra el valor de la métrica devuelta por las distintas configuraciones (F1-Score) en el conjunto de pruebas, por lo que esta información no resulta demasiado reveladora en comparación con la aportada por la pestaña HParams.

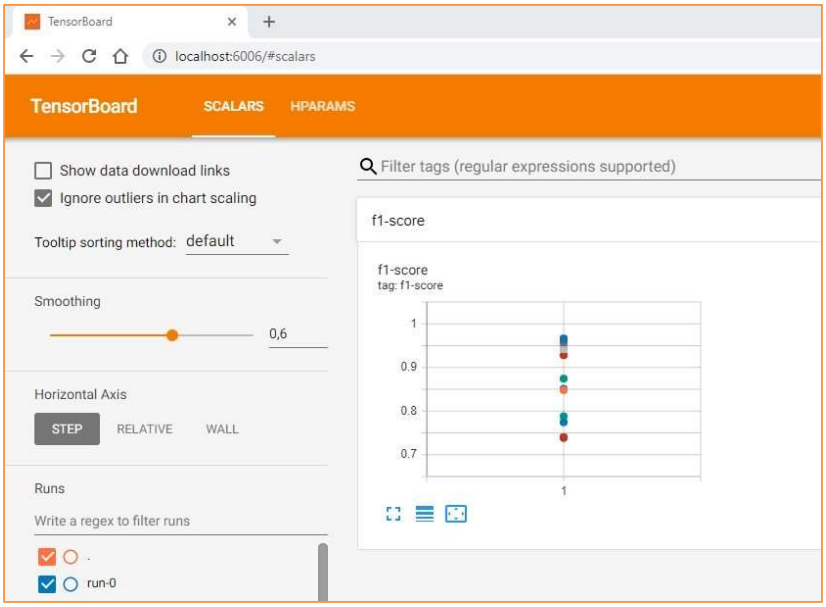
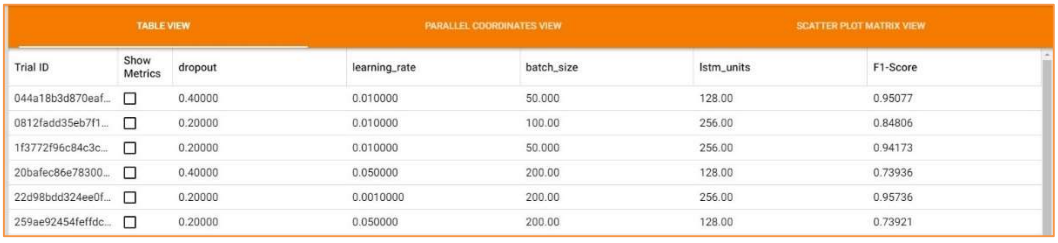


Ilustración 195: Pestaña Scalars

- **HParams:** Esta es la pestaña utilizada para realizar el análisis de la Optimización de hiperparámetros, está compuesta por tres pestañas que permiten visualizar las configuraciones de distintas formas:
    - **Table View (Ilustración 196):** Esta pestaña consiste en una tabla en la que cada fila representa una configuración del modelo y cada columna sus hiperparámetros y métricas correspondientes (en nuestro caso dropout, learning rate, batch size, lstm units y F1-Score).



Trial ID	Show Metrics	dropout	learning_rate	batch_size	lstm_units	F1-Score
044a18b3d870eaf...	☐	0.40000	0.010000	50.000	128.00	0.95077
0812fadd35eb7f1...	☐	0.20000	0.010000	100.00	256.00	0.84806
1f3772f96c84c3c...	☐	0.20000	0.010000	50.000	256.00	0.94173
20bafec86e78300...	☐	0.40000	0.050000	200.00	128.00	0.73936
22d98bdd324ee0f...	☐	0.20000	0.0010000	200.00	256.00	0.95736
259ae92454feffdc...	☐	0.20000	0.050000	200.00	128.00	0.73921

Ilustración 196: Pestaña Table View

- **Parallel Coordinates View (Ilustración 197):** Muestra cada ejecución como una línea que atraviesa un eje por cada hiperparámetro y métrica lo que permite identificar que grupos de hiperparámetros son más importantes.

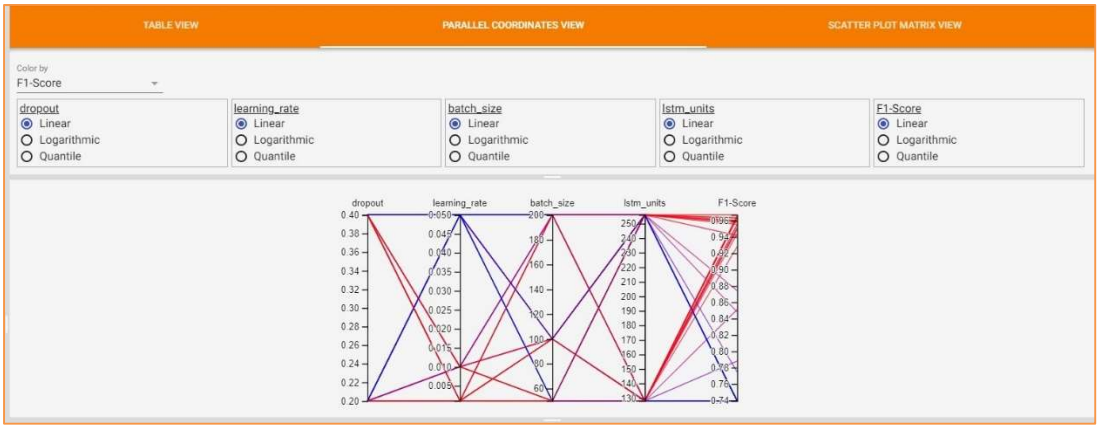


Ilustración 197: Pestaña Parallel Coordinates View

- **Scatter Plot Matrix View (Ilustración 198):** Muestra gráficas representando la relación hiperparámetro/métrica, lo que ayuda a identificar correlaciones.

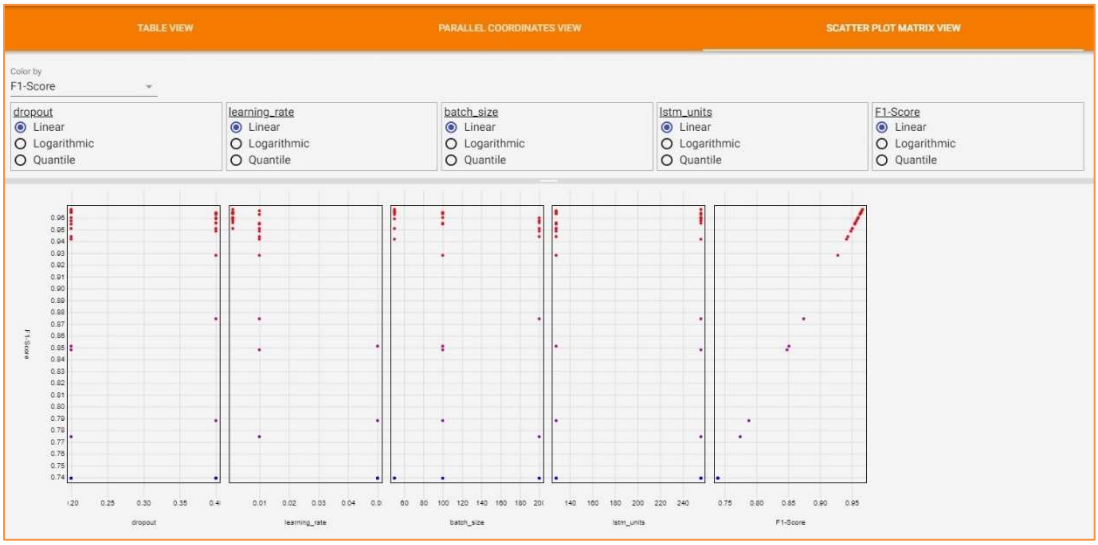


Ilustración 198: Pestaña Scatter Plot Matrix View

En el panel de la izquierda se muestran unos checkbox que nos permiten realizar un filtrado por hiperparámetros, facilitándonos el análisis de configuraciones que cumplen unas condiciones específicas.

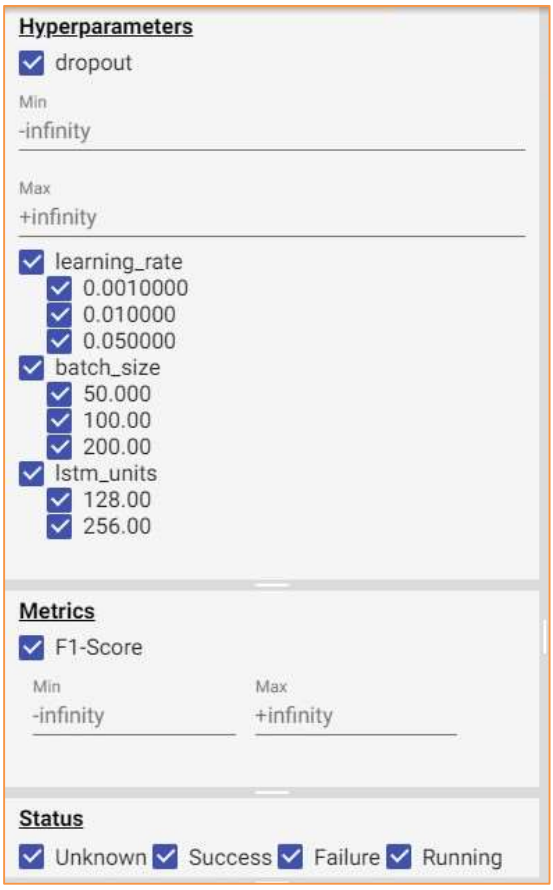


Ilustración 199: Panel de Filtrado por hiperparámetros

PARTE V. CONCLUSIONES

12. Análisis del proceso

Una vez completado el proyecto en los Sprints indicados, se procede a analizar las desviaciones temporales y de costes con respecto a la planificación inicial.

12.1. Desviación de tiempos

En este apartado se realizará un análisis de la desviación temporal del proyecto por Sprints, analizando finalmente la desviación temporal del proyecto de forma global. Aunque todos los Sprints se han completado dentro de los plazos establecidos y sin sufrir ningún tipo de retraso, ha habido varias desviaciones en lo que respecta al tiempo dedicado a cada una de las tareas que los componen.

12.1.1. Sprint 1

El primer Sprint se centró en labores de documentación, análisis y planificación, siendo el preprocesado del conjunto de datos la principal labor de investigación a realizar.

La estimación inicial fue de 40 horas y el Sprint se completó en 46 , por lo que la desviación temporal puede no parecer demasiado significativa. No obstante, tal y como se muestra en la Tabla 65, las tareas de implementación han llevado menos tiempo del esperado, mientras que las de documentación y análisis han requerido de un mayor número de horas de lo que se estimó en un principio.

Esto tendrá una mayor repercusión en la desviación de costes, ya que las tareas de documentación y análisis son realizadas por el Jefe de Proyecto, que tiene un salario mayor que el del Científico de Datos.

Tarea	Rol	Tiempo Estimado	Tiempo Empleado	Desviación
Estudio de Metodologías	Jefe de Proyecto	5h	12h	+7h
Planificación	Jefe de Proyecto	9h	7h	-2h
Preprocesamiento	Científico de Datos	8h	5h	-3h
Reunir conjuntos de datos	Científico de Datos	8h	6h	-2h
Documentación	Jefe de Proyecto	10h	16h	+6h
	TOTAL	**40h**	**46h**	**+6h**

Tabla 65: Desviación temporal Sprint 1

12.1.2. Sprint 2

Durante este Sprint se implementaron los primeros modelos de clasificación. El número total de horas empleadas asciende a 61, siendo solamente una hora más de lo estimado.

La estimación del número de horas totales trabajadas por el Científico de Datos ha sido correcta, aunque de nuevo las tareas de documentación han llevado más tiempo del que se pensó en un principio. Afortunadamente, esto se compensa con una menor cantidad de tiempo dedicado a labores de implementación, por lo que no habrá repercusiones en el coste real por parte del Científico de Datos.

Tarea	Rol	Tiempo Estimado	Tiempo Empleado	Desviación
Documentación	Jefe de Proyecto	10h	11h	+1h
Documentación	Científico de Datos	15h	20h	+5h
Implementación	Científico de Datos	35h	30h	-5h
	TOTAL	**60h**	**61h**	**+1h**

Tabla 66: Desviación temporal Sprint 2

12.1.3. Sprint 3

En este Sprint el número de horas trabajadas ha sido inferior a las estimadas. A pesar de que se hayan completado todas las tareas dentro del plazo establecido, en retrospectiva es fácil reconocer que el número de días asignados a este Sprint (un total de 37) ha sido excesivo para la carga de trabajo establecida.

Inicialmente se pensó que la implementación de la Red Neuronal Recurrente requeriría mucho más tiempo que el resto de los modelos, esta estimación no se ha cumplido gracias a la experiencia adquirida con TensorFlow a lo largo del Sprint 2.

En definitiva, la principal lección aprendida gracias a este Sprint es que se deberían asignado una mayor cantidad de días para la realización del Sprint 4, en el que se ha dispuesto de menos tiempo de lo deseado.

Tarea	Rol	Tiempo Estimado	Tiempo Empleado	Desviación
Documentación	Jefe de Proyecto	10h	13h	+3h
Documentación	Científico de Datos	10h	17h	+7h
Implementación	Científico de Datos	40h	24h	-16h
	TOTAL	**60h**	**54h**	**-6h**

Tabla 67: Desviación temporal Sprint 3

12.1.4. Sprint 4

Durante este Sprint se ha implementado la optimización de hiperparámetros de la Red Neuronal Recurrente, además de realizar la reestructuración del código para permitir su ejecución mediante Scripts de Python.

El número de horas invertidas no se aleja demasiado de las estimadas, aunque el Científico de Datos ha tenido que dedicar más horas de lo que se estimó inicialmente para labores de documentación. Esto se debe principalmente a la redacción de los manuales, que ha llevado más tiempo de lo que se pensó en un principio.

Tarea	Rol	Tiempo Estimado	Tiempo Empleado	Desviación
Documentación	Jefe de Proyecto	20h	16h	-4h
Documentación	Científico de Datos	10h	24h	+14h
Implementación	Científico de Datos	30h	23h	-7h
	TOTAL	**60h**	**63h**	**+3h**

Tabla 68: Desviación temporal Sprint 4

12.1.5. Investigación y reuniones

De las 20 horas estimadas inicialmente, solo han sido necesarias 5, lo que nos deja un margen de 15 horas. Aunque errada, esta cantidad de horas permitirá que la desviación de costes final no sea tan notoria, supliendo en parte las horas excedidas en los Sprints y en Investigación.

En lo que respecta a Análisis e investigación, se estimaron 40 horas (unas diez por Sprint). Finalmente se han empleado un total de 67 horas, esta desviación se debe principalmente a la investigación realizada a lo largo del Sprint 2 para aprender a usar TensorFlow 2.0 y adquirir los conocimientos de procesamiento del lenguaje natural necesarios para la realización del proyecto.

Sprint	Horas Estimadas	Horas Reales	Desviación
Sprint 1	10h	10h	0h
Sprint 2	10h	37h	+27h
Sprint 3	10h	13h	+3h
Sprint 4	10h	7h	-3h
TOTAL	**40h**	**67h**	**+27h**

Tabla 69: Desviación temporal en Investigación

12.1.6. Desviación temporal total del proyecto

En la Tabla 70 se muestra la desviación total del proyecto (a falta de realizar la presentación). Se ha excedido el tiempo total estimado en 16 horas, lo que no supone una desviación demasiado importante con respecto a lo que se planificó en un inicio. No obstante, queda en evidencia que las horas de investigación han superado con creces la estimación, mientras que el tiempo dedicado a reuniones ha sido muy inferior a lo que se pensó en un principio.

Desviación temporal total					
Sprints	**Fecha Inicio**	**Fecha Fin**	**Horas Estimadas**	**Horas Empleadas**	**Desviación**
Sprint 1: Adquisición y preprocesado de datasets	17/01/2020	05/02/2020	40 horas	46 horas	+6 horas
Sprint 2: Aplicación de Baselines	06/02/2020	03/03/2020	60 horas	61 horas	+1 horas
Sprint 3: Implementación de red neuronal recurrente	04/03/2020	10/04/2020	60 horas	54 horas	-6 horas
Sprint 4: Ajuste de parámetros y análisis de los resultados	11/04/2020	30/04/2020	60 horas	63 horas	+3 horas
Total Sprints			**220 horas**	**224 horas**	**+4 horas**
Análisis e investigación			**40 horas**	**67 horas**	**+27 horas**
Reuniones			**20 horas**	**5 horas**	**-15 horas**
TOTAL			**280 horas**	**296 horas**	**+16 horas**

Tabla 70: Desviación temporal total

12.2. Desviación de costes

12.2.1. Costes de personal

Teniendo en cuenta las desviaciones temporales de la sección anterior, procedemos a calcular las desviaciones de costes de personal con respecto a la planificación inicial. El salario por hora de cada trabajador del equipo se muestra en la Tabla 71.

Rol	Euros/Hora
Jefe de Proyecto	19,75
Científico de Datos	11,70

Tabla 71: Costes de Personal por hora

Usando la misma tabla que se empleó en la sección Coste de personal, calculamos las variaciones de costes.

Actividad	Rol	Salario por hora	Duración estimada (Horas)	Coste estimado (euros)	Duración real (Horas)	Coste real (euros)	Desviación (euros)
Análisis e Investigación							
Análisis e Investigación	JP	19,75	0	0	0	0	0
Análisis e Investigación	DS	11,70	40	468	67	783,9	+315,9
TOTAL ACTIVIDAD			40	468	67	783,9	+315,9
Reuniones							
Reuniones	JP	19,75	20	395	5	98,75	-296,25
Reuniones	DS	11,70	0	0	0	0	0
TOTAL ACTIVIDAD			20	395	5	98,75	-296,25
Sprint 1: Adquisición y Preprocesado de Datasets							
Estudio de Metodologías	JP	19,75	5	98,75	12	237	+138,25
Planificación	JP	19,75	9	177,75	7	138,25	-39,5
Documentación	JP	19,75	10	197,5	16	316	+118,5
Preprocesado	DS	11,70	8	93,6	5	58,5	-35,1
Reunir Conjuntos de Datos	DS	11,70	8	93,6	6	70,2	-23,4
TOTAL ACTIVIDAD			40	661,2	46	819,95	+158,75
Sprint 2: Aplicación de Baselines							
Documentación	JP	19,75	10	197,5	11	217,25	+19,75
Documentación	DS	11,70	15	175,5	20	234	58,5
Implementación	DS	11,70	35	409,5	30	351	-58,5
TOTAL ACTIVIDAD			60	782,5	61	802,25	+19,75
Sprint 3: Implementación de Red Neuronal Recurrente							
Documentación	JP	19,75	10	197,5	13	256,75	+59,25
Documentación	DS	11,70	10	117	17	198,9	+81,9
Implementación	DS	11,70	40	468	24	280,8	-187,2
TOTAL ACTIVIDAD			60	782,5	54	736,45	-46,05
Sprint 4: Ajuste de parámetros y análisis de los resultados							
Documentación	JP	19,75	20	395	16	316	-79
Documentación	DS	11,70	10	117	24	280,8	+163,8
Implementación	DS	11,70	30	351	23	269,1	-81,9
TOTAL ACTIVIDAD			60	863	63	865,9	+2,9
TOTAL PROYECTO			**280**	**3952,2**	**296**	**4107,2**	**+155**

Tabla 72: Desviación de costes del proyecto

12.2.2. Costes de material técnico y costes Indirectos

El material se usó de la misma forma que se previó en la sección de planificación, y no ha sido necesario adquirir ningún nuevo Software o servicio, por lo que no se ha producido ninguna desviación en ese sentido. Lo mismo ocurre con los Costes Indirectos, que ya estaban fijados desde el inicio del proyecto.

12.2.3. Desviación de costes total del proyecto

A falta de realizar la presentación, el coste final del proyecto es el mostrado en la siguiente tabla:

Tipo	Coste (€)
Coste de Personal	4107,2
Material técnico	569,53
Ganancias y Beneficios	983,35
Impuesto de Sociedades	245,84
TOTAL	5905,29

Tabla 73: Coste final del proyecto

El presupuesto establecido inicialmente para el proyecto se ha excedido en un total de 155 euros.

12.3. Lecciones aprendidas

- La estimación temporal de tareas que incluyen el uso de tecnologías sobre las que no se tiene conocimiento es una tarea complicada y en la que es común cometer errores. A lo largo de este proyecto, se ha descubierto que es recomendable asignar una gran cantidad de horas al aprendizaje de la tecnología en cuestión. Esto se refleja especialmente en la desviación en el número de horas dedicadas a investigación durante el Sprint 2.

  Sin duda una mayor experiencia con las tecnologías utilizadas habría ahorrado desviaciones innecesarias. Aun así, la desviación no ha sido demasiado notoria y se han podido cumplir todos los objetivos establecidos para cada Sprint.
- Para futuros proyectos, se asignará una mayor cantidad de horas a las labores de documentación, ya que en todos los Sprints se ha superado la estimación inicial.

En definitiva, gestionar un proyecto con unos plazos definidos para su desarrollo es una tarea de una complejidad muy elevada y que requiere grandes dosis de motivación, dedicación continua y un buen seguimiento para su correcta realización.

13. Conclusiones

13.1. Consecución de los objetivos del proyecto

Con esta sección llegamos al final del proyecto. Se han cumplido todos los requisitos establecidos, además de los objetivos docentes y técnicos.

El resultado de este proyecto consiste en un total de 9 modelos de clasificación de textos distintos (aunque muchos de ellos también tienen aplicaciones en otros ámbitos de Inteligencia Artificial y Deep Learning), además de un análisis comparativo entre estos, analizando las ventajas e inconvenientes de cada. También se ha llevado a cabo la optimización de los hiperparámetros de la Red Neuronal Recurrente.

Obteniendo un valor de F1-Score del 96.66% en el conjunto de pruebas de la Red Neuronal Recurrente, se han superado los resultados de la publicación original empleando solamente la información del texto. Esta mejora de los resultados se debe principalmente a un Preprocesado exhaustivo del dataset y a un filtrado de los datos (eliminación de ruido), como en el caso de las noticias escritas en idiomas distintos al inglés.

A diferencia de lo que ocurre en muchos proyectos de Machine Learning, en este proyecto se ha decidido comenzar aplicando métodos más simples para familiarizarnos con el conjunto de datos y detectar posibles sesgos, en lugar de empezar directamente aplicando los métodos más complejos que existen en la actualidad. Este enfoque también ha sido útil a la hora de aprender a usar las librerías requeridas para el proyecto, implementando modelos cuya complejidad va en aumento, lo que ha generado una curva de aprendizaje idónea. Las tecnologías aprendidas gracias a este desarrollo están entre las más demandadas en el mercado profesional en la actualidad.

Los sesgos detectados en el conjunto de datos referentes al vocabulario de las elecciones y a la longitud de las noticias explican en gran medida los buenos resultados del artículo original. Una de las críticas más comunes entre los detractores de los modelos neuronales es que se trata de modelos de “caja negra” que reciben una serie de entradas y devuelven un conjunto de salidas, sin obtener ninguna explicación del porqué de dicha clasificación ni permitir realizar modificaciones (algo que sí ocurre en otros modelos, como los árboles de clasificación).Puede ocurrir que un modelo neuronal devuelva muy buenos resultados tanto en entrenamiento como en pruebas, pero no esté prediciendo lo que se cree. Esta clase de clasificadores se conocen como “False predictors”, y sus causas suelen ser principalmente que la clase a predecir se encuentra dentro de los parámetros de entrada o que existen una gran cantidad de duplicados en el conjunto de entrenamiento y pruebas.

En este caso no llega a tratarse de un False Predictor, pero gracias a este proyecto se ha demostrado que los resultados no son tan buenos como anuncian las cifras. Por lo tanto, una de las principales lecciones aprendidas gracias a este desarrollo ha sido que es conveniente no comenzar aplicando los modelos más complejos a nuestra disposición, ya que hay muchas ocasiones en los que los más simples son más que suficientes, además de permitirnos averiguar si la calidad del conjunto de datos es tan buena como puede aparentar en un principio.

El desarrollo de este trabajo también ha servido para poder emplear metodologías ágiles de forma práctica en un proyecto de cierta envergadura. Scrum es una metodología de desarrollo que ha sido explicada y enseñada numerosas veces a lo largo del grado. Sin embargo, este proyecto ha supuesto la primera oportunidad de ponerlo a prueba de una manera muy similar a la que podríamos aplicar en el mundo real. Esta metodología nos ha permitido aprender a desarrollar de forma más eficiente mediante iteraciones de corta duración, imponiendo retos razonables con la cantidad de tiempo disponible y realizando un seguimiento continuo del progreso realizado.

Por último, este proyecto ha servido como guía y preparación para enfrentar a problemas de este ámbito en el mundo real, además de haber comprendido los conceptos matemáticos que hay detrás de toda la implementación realizada.

13.2. Trabajo futuro

A pesar de haber llegado al final del desarrollo cumpliendo con todos los objetivos establecidos, todavía pueden realizarse propuestas de trabajo futuro interesantes, como ocurre en la mayoría de los proyectos. Algunas líneas de trabajo que podrían resultar prometedoras son las siguientes:

- **Más datos:** Este suele ser un elemento común en la sección de Trabajo futuro de proyectos de Machine Learning. Una mayor cantidad de datos perteneciente a nuevos contextos informativos permitiría que el modelo pudiese detectar más patrones característicos de las fake news, además de albergar información sobre dichos contextos para contrastar (Siempre que los datos añadidos sean de calidad, hayan sido correctamente etiquetados y se eviten sesgos). La estructura de la red actual podría ser entrenada con una mayor cantidad de información sin necesidad de realizar modificaciones importantes.
- **Despliegue utilizando TensorFlow Serving:** Esta mejora permitiría darles un uso práctico a las redes implementadas, generando una API que reciba como parámetro un texto y devuelva la categoría a la que pertenece. TensorFlow Serving ofrece muchas ventajas con respecto a otras tecnologías de despliegue como Flask, al ser altamente escalable y permitir entrenar los modelos de forma continua sin que sea necesario volver a desplegar. También permite tener almacenados varios modelos en el servidor de manera simultánea, pudiendo elegir en todo momento cual se va a emplear, esta opción aporta muchas posibilidades, ya que permitiría tener un modelo estable ofreciendo un servicio mientras se entrena otro con una mayor cantidad de datos o una estructura diferente, pudiendo volver siempre al modelo anterior en el caso de que la nueva versión tenga un rendimiento inferior al esperado.
- **Desarrollo de una extensión para navegador:** Esta mejora sería un incremento muy interesante de la expuesta en el punto anterior. Consistiría en desarrollar una extensión para navegadores que, empleando técnicas de Web Scraping, extrajera el contenido de la página en la que se encuentra el usuario (asumimos que solo realizaría esta acción si

se trata de una web de noticias), mandase una petición a la API e informase al usuario en el caso de que la noticia fuese potencialmente falsa.

BIBLIOGRAFÍA Y REFERENCIAS

[1] «TI-CNN: Convolutional Neural Networks for Fake News Detection,» [En línea]. Available: https://arxiv.org/abs/1806.00749.

[2] «Significado de Clickbait,» [En línea]. Available: https://www.significados.com/clickbait/.

[3] «Without the Russians, Trump wouldn't have won,» [En línea]. Available: https://www.washingtonpost.com/opinions/without-the-russians-trump-wouldnt-have-won/2018/07/24/f4c87894-8f6b-11e8-bcd5-9d911c784c38_story.html?utm_term=.ac812004f1e2.

[4] «The 7 Steps of Machine Learning (AI Adventures),» [En línea]. Available: https://www.youtube.com/watch?v=nKW8Ndu7Mjw.

[5] «Costes Directos,» [En línea]. Available: https://economipedia.com/definiciones/coste-directo.html.

[6] «Boletín Oficial del Estado,» [En línea]. Available: https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf.

[7] «Bases y tipos de cotización 2019,» [En línea]. Available: http://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537.

[8] «Método de Amortización Lineal,» [En línea]. Available: https://www.supercontable.com/informacion/Contabilidad/Metodo_de_Amortizacion_Lineal.html.

[9] «Licencias Windows 10,» [En línea]. Available: https://www.microsoft.com/es-es/store/b/windows.

[10] «Tienda Microsoft Office 365,» [En línea]. Available: https://www.microsoft.com/es-es/microsoft-365/business/compare-more-office-365-for-business-plans?rtc=1&ef_id=EAIaIQobChMIlMma9Yad5wIV04jVCh0VGg6hEAAYAiAAEgLSGPD_BwE:G:s&ef_id=EAIaIQobChMIlMma9Yad5wIV04jVCh0VGg6hEAAYAiAAEgLSGPD_BwE:G:s&OCID=AID2000748_SEM.

[11] «Impuesto sobre sociedades,» [En línea]. Available: https://es.wikipedia.org/wiki/Impuesto_sobre_sociedades.

[12] «Traditional vs. Agile Software Development Methodologies,» [En línea]. Available: http://www.kpipartners.com/blog/traditional-vs-agile-software-development-methodologies.

[13] «Metodologías del Desarrollo de Software,» [En línea]. Available: https://okhosting.com/blog/metodologias-del-desarrollo-de-software/#Que_es_una_Metodologia.

[14] «Manifiesto por el desarrollo Ágil de Software,» [En línea]. Available: https://agilemanifesto.org/iso/es/manifesto.html.

[15] «Manifiesto Ágil,» [En línea]. Available: https://es.wikipedia.org/wiki/Manifiesto_%C3%A1gil.

[16] «Metodologías... ¿Tradicional vs Ágil?,» [En línea]. Available: https://tech.tribalyte.eu/blog-metodologias-tradicional-vs-agil.

[17] «Metodología De Desarrollo: Tradicional vs Ágil,» [En línea]. Available: http://noticiaseinformaciondetic.blogspot.com/2018/08/metodologia-de-desarrollo-tradicional.html.

[18] «Extreme Programming,» [En línea]. Available: https://www.agilealliance.org/glossary/xp/#q=~(infinite~false~filters~(postType~(~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'xp))~searchTerm~'~sort~false~sortDirection~'asc~page~1).

[19] «Funcionamiento de la Metodología XP,» [En línea]. Available: https://sites.google.com/site/xpmetodologia/marco-teorico/funcionamiento.

[20] «Qué es la metodología Kanban y cómo utilizarla,» [En línea]. Available: https://www.iebschool.com/blog/metodologia-kanban-agile-scrum/.

[21] «Kanban o cómo ser flexibles, pero poniendo límites,» [En línea]. Available: https://qanewsblog.com/2016/05/11/kanban-o-como-ser-flexibles-pero-poniendo-limites/.

[22] «Feature Driven Development (FDD) and Agile Modeling,» [En línea]. Available: http://agilemodeling.com/essays/fdd.htm.

[23] «METODO AGIL ASD,» [En línea]. Available: http://ingenieriadesoftware.mex.tl/61154_asd.html.

[24] «Deconstruyendo SCRUM,» [En línea]. Available: https://www.deconstruyendoscrum.com/sabias-que-scrum-va-de-cerdos-y-pollos/.

[25] «What is Burndown chart in Scrum?,» [En línea]. Available: https://www.visual-paradigm.com/scrum/scrum-burndown-chart/.

[26] «Taiga,» [En línea]. Available: https://taiga.io.

[27] «Toggl,» [En línea]. Available: https://toggl.com.

[28] «PyCharm,» [En línea]. Available: https://www.jetbrains.com/es-es/pycharm/.

[29] «Apache License,» [En línea]. Available: https://www.apache.org/licenses/LICENSE-2.0.

[30] «Documentación de Jupyter Notebook,» [En línea]. Available: https://jupyter-notebook.readthedocs.io/en/stable/.

[31] «Documentación de TensorFlow,» [En línea]. Available: https://www.tensorflow.org.

[32] «Which Deep Learning Framework is Growing Fastest?,» [En línea]. Available: https://towardsdatascience.com/which-deep-learning-framework-is-growing-fastest-3f77f14aa318.

[33] «Documentación pandas,» [En línea]. Available: https://pandas.pydata.org.

[34] «Documentación Numpy,» [En línea]. Available: https://numpy.org/doc/.

[35] «nltk documentation,» [En línea]. Available: https://www.nltk.org.

[36] «Sklearn Documentation,» [En línea]. Available: https://scikit-learn.org.

[37] «Gensim Documentation,» [En línea]. Available: https://radimrehurek.com/gensim/.

[38] «Kaggle,» [En línea]. Available: https://www.kaggle.com.

[39] «What is Web Scraping and What is it Used For?,» [En línea]. Available: https://www.parsehub.com/blog/what-is-web-scraping/.

[40] «How popular are neural networks over the years?,» [En línea]. Available: https://www.lucidarme.me/popular-neural-networks-years/.

[41] «Everything you need to know about Neural Networks and Backpropagation,» [En línea]. Available: https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a.

[42] «Artificial Neural Network - Perceptron,» [En línea]. Available: https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/.

[43] «Función sigmoide,» [En línea]. Available: https://es.wikipedia.org/wiki/Funci%C3%B3n_sigmoide.

[44] «A Gentle Introduction to Exploding Gradients in Neural Networks,» [En línea]. Available: https://machinelearningmastery.com/exploding-gradients-in-neural-networks/.

[45] «How the backpropagation algorithm works,» [En línea]. Available: http://neuralnetworksanddeeplearning.com/chap2.html.

[46] «Propagación hacia atrás,» [En línea]. Available: https://es.wikipedia.org/wiki/Propagaci%C3%B3n_hacia_atr%C3%A1s.

[47] «Binary crossentropy loss function,» [En línea]. Available: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/binary-crossentropy.

[48] «What is the difference between a parameter and an hyperparameter,» [En línea]. Available: https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/.

[49] «Dropout in (Deep) Machine learning,» [En línea]. Available: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5.

[50] «Overfitting and Underfitting With Machine Learning Algorithms,» [En línea]. Available: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/.

[51] «Precision, Recall, F1, Accuracy en clasificación,» [En línea]. Available: https://iartificial.net/precision-recall-f1-accuracy-en-clasificacion/.

[52] «Functional Programming,» [En línea]. Available: https://en.wikipedia.org/wiki/Functional_programming.

[53] «TI-CNN dataset,» [En línea]. Available: https://drive.google.com/file/d/0B3e3qZpPtccsMFo5bk9Ib3VCc2c/view.

[54] «Getting Real About Fake News (Kaggle Dataset),» [En línea]. Available: https://www.kaggle.com/mrisdal/fake-news.

[55] «Fake news detection using Deep Learning,» [En línea]. Available: https://www.researchgate.net/publication/336361285_Fake_news_detection_using_Deep_Learning.

[56] «Artículos en los que el artículo original es citado,» [En línea]. Available: https://scholar.google.es/scholar?hl=es&as_sdt=2005&sciodt=0,5&cites=7199464375614139 67&scipsc=&q=.

[57] «Pre-Processing in Natural Language Machine Learning,» [En línea]. Available: https://towardsdatascience.com/pre-processing-in-natural-language-machine-learning-898a84b8bd47.

[58] «Stemming and Lemmatization,» [En línea]. Available: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html.

[59] «WordNet,» [En línea]. Available: https://wordnet.princeton.edu.

[60] «Always start with a stupid model, no exceptions,» [En línea]. Available: https://blog.insightdatascience.com/always-start-with-a-stupid-model-no-exceptions-3a22314b9aaa.

[61] «Bag-of-words model,» [En línea]. Available: https://en.wikipedia.org/wiki/Bag-of-words_model.

[62] «A Gentle Introduction to the Bag-of-Words Model,» [En línea]. Available: https://machinelearningmastery.com/gentle-introduction-bag-words-model/.

[63] «Text Analytics – Term Frequency,» [En línea]. Available: https://www.darrinbishop.com/blog/2017/11/text-analytics-term-frequency/.

[64] «Understanding Inverse Document Frequency: On theoretical arguments for IDF,» [En línea]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.438.2284&rep=rep1&type=pdf.

[65] «tfidf,» [En línea]. Available: http://www.tfidf.com/.

[66] «Naive Bayes (sklearn),» [En línea]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html.

[67] «All About Naive Bayes,» [En línea]. Available: https://towardsdatascience.com/all-about-naive-bayes-8e13cef044cf.

[68] «Efficient Estimation of Word Representations inVector Space,» [En línea]. Available: https://arxiv.org/pdf/1301.3781.pdf.

[69] «Distributed Representations of Words and Phrasesand their Compositionality,» [En línea]. Available: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[70] «NLP from Zero to One: for classification and machine translation: Part One,» [En línea]. Available: https://mc.ai/nlp-from-zero-to-one-for-classification-and-machine-translation-part-one/.

[71] «Distributed Representations of Sentences and Documents,» [En línea]. Available: https://arxiv.org/pdf/1405.4053.pdf.

[72] «CountVectorizer (Documentación Sklearn),» [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html.

[73] «TfIdfVectorizer (Documentación de Sklearn),» [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

[74] «Word2Vec (Documentación gensim),» [En línea]. Available: https://radimrehurek.com/gensim/models/word2vec.html.

[75] «Doc2vec paragraph embeddings,» [En línea]. Available: https://radimrehurek.com/gensim/models/doc2vec.html.

[76] «On the difficulty of training recurrent neural networks,» [En línea]. Available: http://proceedings.mlr.press/v28/pascanu13.pdf.

[77] «Understanding LSTM Networks,» [En línea]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[78] «OVERVIEW OF RECURRENT NEURAL NETWORKS AND THEIR APPLICATIONS,» [En línea]. Available: https://analyticsindiamag.com/overview-of-recurrent-neural-networks-and-their-applications/.

[79] «LONG SHORT-TERM MEMORY,» [En línea]. Available: http://www.bioinf.jku.at/publications/older/2604.pdf.

[80] «Understanding LSTM and its diagrams,» [En línea]. Available: https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714.

[81] «Tokenizer de TensorFlow,» [En línea]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer.

[82] «TensorFlow Bidirectional Layer,» [En línea]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Bidirectional.

[83] «TensorFlow 2.0 LSTM Layer,» [En línea]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM.